目　　录



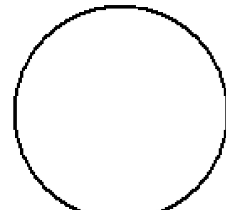








第1章 CACSD 软件环境与新技术概述



1.1 绪　论

随着科学技术的发展，控制理论和系统的研究显得越来越重要。在 40 年代控制理论作为一门独立的学科出现以来，已经得到了迅速的发展。开始时控制系统的设计可以由纸笔等工具容易地计算出来，如 Ziegler 与 Nichols 于 1942 年提出的 PID 经验公式就可以十分容易地设计出来。随着控制理论的迅速发展，控制的效果要求得越来越高，控制算法越来越复杂，控制器的设计也越来越困难，这样光利用纸笔以及计算器等简单的运算工具难以达到预期的效果，加之在计算机领域取得了迅速的发展，于是很自然地出现了控制系统的计算机辅助设计 (Computer-Aided Control System Design，简称为 CACSD) 方法。

控制系统的计算机辅助设计技术的发展目前已达到了相当高的水平，并一直受到控制界的普遍重视。1982 年 12 月和 1984 年 12 月，控制系统领域在国际上最权威的 IEEE 控制系统学会 (Control Systems Society, 简称为 CSS) 的控制系统杂志 (Control Systems Magazine) 和 IEEE 学会的科研报告 (Proceedings of IEEE) 分别第一次出版了关于 CACSD 的专刊[9, 10]，美国的著名学者 Jamshidi 与 Herget 分别于 1985 年和 1992 年出版了两本著作来展示 CACSD 领域的最新进展[11, 12]。在如国际自动控制联合会世界大会 (IFAC World Congress)、美国控制会议 (American Control Conference, 简称为 ACC) 及 IEEE 的控制与决策会议 (Conference on Control and Decision, 简称为 CDC) 等各种国际控制界的重要学术会议上都有有关 CACSD 的专题会议及各种研讨会，可见该领域的发展是异常迅速的，控制系统计算机辅助设计又常常称作计算机辅助控制系统工程 (Computer-Aided Control Systems Engineering, 简称 CACSE)。

近 30 年来，随着计算机技术的飞速发展，出现了很多的优秀计算机应用软件，在控制系统的计算机辅助设计领域更是如此，各类 CACSD 软件频繁出现且种类繁多，有的是用 FORTRAN 语言编写的软件包，有的是人机交互式软件系统，还有专用的仿真语言，在国际控制界广泛使用的这类软件就有几十种之多。

在国内流行的大多数有关教材中，控制系统计算机辅助设计主要讲述"经典"的 CACSD 方法，如串联校正、简单的二次型最优控制策略等，而对当前国际上比较热门的 CACSD 方法诸如自整定 (autotuning) PID 控制方法、定量反馈理论 (quantitative feedback theory, 简称为 QFT) 方法、以及 H_∞ 鲁棒控制方法等并无系统的介绍。本书的目的之一就是系统地引入这些先进的方法，使得读者能更开阔自己的眼界。



以前国内外在介绍控制理论及 CACSD 技术的教材中，都采用 BASIC 语言 [13, 28] 和 FORTRAN 语言 [18]，并有少部分教材中采用 C 语言作为主要的程序设计语言。这固然可以锻炼学生，使之可以从最底层了解有关的 CACSD 程序设计的方法与技巧，但难以使之能有一个对整个方法的全面了解，容易产生“只见树木，不见森林”的认识偏差。另外由于程序往往是用户自己编写的，所以难免会造成数值不稳定、计算结果错误等不该发生事件的出现，此外，这种从最底层进行编程的方式在效率上来讲是相当低的，大部分时间将花费在没有太大价值的重复性机械性劳动上了。

孟子云：“工欲善其事，必先利其器”。跟踪国际最先进的 CACSD 软件环境及发展，以当前国际上最流行的 CACSD 软件环境 MATLAB 为基本出发点来系统地介绍控制系统计算机辅助设计技术及软件实现，从而大大提高 CACSD 算法研究与实际应用的效率和可靠性，这也是本书的另外一个目的。

1.2 国外 CACSD 软件环境综述

1973 年美国学者 Melsa 教授和 Jones 博士出版了一本专著 [18]，书中给出了许多当时流行的控制系统计算机辅助分析与设计的程序，包括求取系统的根轨迹、频率响应、时间响应、以及各种控制系统设计的子程序如 Luenberger 观测器、Kalman 滤波等。瑞典 Lund 工学院 Karl Åström 教授主持开发的一套交互式 CACSD 软件 INTRAC (IDPAC, MODPAC, SYNPAC, POLPAC 等，以及仿真语言 SIMNON) [2]，其中的 SIMNON 仿真语言要求用户依照它所提供的语句编写一个描述系统的程序，然后才可以对控制系统进行仿真。日本的古田胜久 (Katsuhisa Furuta) 教授主持开发的 DPACS-F 软件 [6]，在处理多变量系统的分析和设计上还是很有特色的。在国际上流行的仿真语言 ACSL, CSMP, TSIM, ESL 等也同样要求用户编写模型程序，并提供了大量的模型模块。在这一阶段还出现了很多的专用程序，如英国剑桥大学推出的线性系统分析与设计软件 CLADP (Cambridge linear analysis and design programs)[5, 17] 与美国国家宇航局 (NASA) Langley 研究中心的 Armstrong 开发的 LQ 控制器设计的 ORACLS (optimal regulator algorithms for the control of linear systems)[1] 等。

1980 年美国学者 Cleve Moler 等人推出的交互式 MATLAB 语言逐渐受到了控制界研究者的普遍重视，从而陆续出现了许多专门用于控制理论及其 CAD 的工具箱。图形交互式的模型输入计算机仿真环境 SIMULINK 的出现为 MATLAB 应用的进一步推广起到了积极性的推动作用。现在，MATLAB 已经风靡了全世界，成为控制系统 CAD 领域最普及也是最受欢迎的软件环境。

在 MATLAB 迅速发展的同时，很多软件开发者针对控制系统领域的实际问题开发了专用的 CACSD 计算机辅助设计软件，如美国系统控制技术公司 (Systems Control Technology, Inc) 的 John Little 等人研制的 CTRL-C, Boeing 公司的 EASY 5 及 EASY5x, Integrated.Systems 公司的 MATRIXx 及 Xmath，Systems Technology Incorporated 公司的 CC 程序，Visual Simulation 公司的 VisSim, 日本东京工学院的 MaTX 等，其中很多软件是并行于 MATLAB 而独立开发的，但或多或少都会从这些软件的语句结构或使



用方法中看出受到 MATLAB 影响的痕迹，所以说，从国际上最流行的 MATLAB 出发来介绍控制系统的计算机辅助设计技术是再合适也不过的了。这就是本书在众多 CACSD 软件中挑选 MATLAB 作为基本语言的一个最主要的原因。

国际上控制系统计算机辅助设计软件的发展大致分为几个阶段：软件包阶段、交互式语言阶段及当前的面向对象的程序环境阶段 [14] 。

在早期的工作中， CACSD 主要集中在软件包的编写上，如前面提及的 Melsa 和 Jones 的著作。从数值算法的角度上也出现了一些著名的软件包，如美国的基于特征值的软件包 EISPACK[7, 26] 和线性代数软件包 LINPACK[4]，英国牛津数值算法研究组 (Numerical Algorithm Group) 开发的 NAG 软件包 [21] 及文献 [23] 中给出的程序集等，在 CACSD 领域的经典软件包作品有英国 Kingston Polytechnic 控制系统研究组开发的 SLICE (subroutine library in control engineering) 软件包，前面提及的 DPACS-F, ORACLS 等。这些软件包大都是由 FORTRAN 语言编写的源程序组成的，例如若想求出 N 阶实矩阵 A 的全部特征值 (用 `WR`，`WI` 数组分别表示其实虚部) 和对应的特征向量矩阵 Z，则 EISPACK 软件包给出的子程序建议调用路径为

```
CALL BALANC(NM,N,A,IS1,IS2,FV1)
CALL ELMHES(NM,N,IS1,IS2,A,IV1)
CALL ELTRAN(NM,N,IS1,IS2,A,IV1,Z)
CALL HQR2(NM,N,IS1,IS2,A,WR,WI,Z,IERR)
IF (IERR.EQ.0) GOTO 99999
CALL BALBAK(NM,N,IS1,IS2,FV1,N,Z)
```

由上面的叙述可以看出，要求取矩阵的特征值和特征向量，首先要给一些数组和变量依据 EISPACK 的格式作出定义和赋值，并编写出主程序，再经过编译和连接过程，形成可执行文件，最后才能得出所需的结果。用软件包的形式来编写程序有如下的缺点：

- **使用不方便**: 对不是很熟悉 EISPACK 的用户来说，直接利用软件包来编写程序是相当困难的，也是相当容易出错的，其中一个子程序调用发生微小的错误可能将导致最终得出错误的结果。
- **调用过程繁琐**: 首先需要编写主程序，确定对软件包的调用过程，再经过必要的编译和连接过程，有时还要花大量的时间去调试程序以保证其正确性，而不是想得出什么马上就可以得出的。
- **执行程序过多**: 想求解一个特定的问题就需要编写一个专门的程序，并形成一个可执行文件，如果需要求解的问题很多，就需要在计算机硬盘上同时保留很多这样的可执行文件，这样计算机磁盘空间的利用不是很经济。
- **不利于传递数据**: 通过软件包调用方式针对每个具体问题就能形成一个孤立的可执行文件，在一个程序中产生的数据无法传入另一个程序，更无法使几个程序同时执行来解决所关心的问题。
- **维数指定困难**: 在 CACSD 中最重要的变量是矩阵，如果要求解的问题维数较低，则形成的程序就不能用于求解高阶问题，例如文献 [18] 中的程序均定为 10 阶。所以有



时为使得程序通用，往往将维数设置得很大，这样在解小规模问题是会出现空间的浪费，而大规模问题仍然求解不了。在优秀的软件中往往需要动态地进行矩阵定维。

此外，这里介绍的大多数早期软件包都是由 FORTRAN 语言编写的，由于众所周知的原因，以前 FORTRAN 语言绘图并不是轻而易举的事情，这就需要再调用相应的软件包来做进一步处理，在绘图方面比较实用和流行的软件包是 GINO-F[3]，但这种软件包只给出绘图的基本子程序，所以要绘制较满意的图形需要用户自己去用这些低级命令去编写出合适的绘图子程序来。

英国剑桥大学学者 John Edmunds 和 Jan Maciejowski 等人开发的 CLAPD 在控制界享有盛誉，它包括了多变量系统分析与设计的多种方法，其中有 Nyquist 类以及特征轨迹等多变量频域设计方法，也有 LQG 与 Kalman 滤波等时域设计方法，还可以处理时间延迟及分布系统等非有理问题。日本东京工学院的古田胜久教授主持开发的 DPACS-F 软件是由 FORTRAN 语言编写的，它可以由状态空间和频域方法来分析多变量线性系统，并可以由极点配置和 LQG 等方法来设计控制器，此外还可以进行多变量系统辨识等工作。NASA 的 Armstrong 教授编写的 ORACLS 则是一个十分专用的软件，它可以用于多变量系统的 LQG 设计，该软件也是由 FORTRAN 语言编写的。

70 年代末期和 80 年代初期出现了很多实用的具有良好人机交互功能的软件，MATLAB 就是其中的一个成功的范例，此外前面提及的 INTRAC 和 CTRL-C 等也是优秀的人机交互式软件，例如在 CTRL-C 下用户可以输入下面的命令来给出矩阵参数并求取该矩阵的特征值与特征向量

```
[>  A=[1,2,3; 4,5,6; 7,8,0];
[>  [x,d]=eig(A)
```

这里 [> 为 CTRL-C 的专用提示符，在这一提示符下可以容易地输入 A 矩阵，并由一条命令直接求出 A 矩阵的特征值和特征向量，这无疑使得问题的解法得到了极大的简化。这种输入方式和后面将介绍的 MATLAB 几乎是一致的，另外若已经获得了数据，则利用 CTRL-C 可以容易地绘制出相应的曲线来。可以看出，使用 CTRL-C 这样的交互式软件系统比起软件包来说上了一个台阶。交互式软件的一个显著特点是其使用方便、集成度高，由简单的几条规范命令就可以实现以前 FORTRAN 类语言成百上千条语句的功能，且结果稳定可靠。

正因为存在多种多样的 CACSD 软件，而它们之间又各有所长，所以在 CACSD 技术的发展过程中曾有过几次将若干常用软件集成在一起的尝试，例如 1984 年前后美国学者 Spang, III 教授曾将当时流行的 SIMNON, CLAPD, IDPAC 及他自己研制的 SSDP(state space design program) 集成在一起，形成了一个强大的软件 [27]，各个组成软件之间是靠读写文件的方式来传递数据的，这多少可以解决前面提及的程序之间不能传递数据的弊病。1986 年前后由英国科学与工程委员会 (SERC) 资助，Harold Rosenbrock 教授和 Neil Munro 教授主持的，英国多所大学和研究机构参与的 ECSTASY (environment for control system theory and synthesis) 软件环境的开发项目 [20]，在该软件中试图将流行的新一代软件如 MATLAB, ACSL, TSIM 甚至新近出现的 Mathematica 等集成到一个框



架之下，该软件还可以同时自动采用 LaTeX和 Framemaker 等来输出专业的排版结果，并取得了一些成效。各个软件之间的数据传递是通过数据库来实现的，ECSTASY 定义了方便实用的 CACSD 新命令，比 MATLAB 更为简洁。ECSTASY 这样的软件是一个有益的尝试，但该软件当时只可以在 Sun 工作站上运行，并没有考虑 PC 机的兼容性。

依作者之见，这些集成出来的软件并不是很成功的，因为它们并没有达到预期的效果，事实上，从那以后每个软件的功能都有了明显的改善，MATLAB 语言有了自己的仿真功能，SIMULINK 从某种意义上来讲其功能和接口更优于 ACSL, MATLAB 和 Mathematica 之间也有了较好的接口，它们的优势可以得到充分地互补。

1.3 ACSL 仿真语言及其应用

ACSL 是一类有代表性意义的仿真语言，它是由美国 Mitchell and Gauthier Associate 公司推出的仿真语言，其全名为 Advanced Continuous Simulation Language (高级连续仿真语言) [19]。ACSL 的语句规则是建立在 1967 年由仿真委员会 (Simulation Councils Inc, 简称为 SCi) 规定的仿真语言规范的基础上的。

ACSL 要求用户用它所提供的模块编写一个描述系统的程序，这一程序和 FORTRAN 语言的程序十分相似，其显式结构如表 1-1 所示。ACSL 首先要求用户依照其语言规

表 1-1 ACSL 仿真语言的基本程序结构

PROGRAM	程序名称
INITIAL	
	在运行之前要执行的语句在一般情况下为初始条件等的定义。
END	
DYNAMIC	
DERIVATIVE	
	用积分语句 INTEG 描述的微分方程计算的语句
END	
	其它计算语句。
END	
TERMINAL	
	终止判断语句，通常在 TERMT 语句条件为真时终止
END	

则建立起一个模型文件，然后可以通过 ACSL 本身提供的命令对之进行仿真及辅助分析。ACSL 与 FORTRAN 语言的主要区别在于，ACSL 的语句更简练，内容更丰富，ACSL 语言也可以直接调用由 FORTRAN 编写的子程序。ACSL 编程的结构比相应的 FORTRAN 语言更严格。程序的基本结构必须严格按表 1-1 所给出的格式来编写，否则所得出的仿真结果可能出现意想不到的错误。ACSL 提供了几十个系统子模块 (macros)，其中包括很常用的线性和非线性子模块，如传递函数模块 TRAN, 积分器模块 INTEG, 超前滞后环节 LEDLAG, 延迟模块 DELAY, 死区非线性模块 DEAD, 磁滞回环 BAKLSH, 限幅积分器 LIMINT 等，用户可以利用这些子模块简单地编写出描述给定系统的仿真模型，然后采用 ACSL 提供的功能来对系统进行仿真分析，并绘制出结果的曲线表示。例如，著名的



Van der Pol 方程的一种特殊形式可以写成

$$\ddot{y}+\mu(y^2-1)\dot{y}+y=0$$

若取 $\mu=1$, 并取状态变量为 $y_1=y, y_2=\dot{y}$，则 Van der Pol 方程可以写成

$$\dot{y}_1=y_1(1-y_2^2)-y_2,\ \dot{y}_2=y_1$$

这时可以由 ACSL 所定义的语言写出此系统的模型如下：

```
PROGRAM VAN DER POL EQUATION
CINTERVAL CINT=0.01
CONSTANT  Y1C=3.0,   Y2C=2.5,   TSTP=15.0
Y1=INTEG(Y1*(1-Y2**2)-Y2, Y1C)
Y2=INTEG(Y1, Y2C)
TERMT (T.GE.TSTP)
END
```

在此程序中把显示步长 CINT 设置为 0.01, 状态变量的初值由 Y1C 和 Y2C 表示，而终止仿真时间 TSTP 设置为 15，该程序中变量 T 为实际仿真时间。这时可以采用 ACSL 的编译命令来编译并将此模型和 ACSL 库连接起来，则可以形成一个可执行文件。这一过程完成之后 ACSL 将自动给出提示：

```
ACSL>
```

用户可以在此提示符下键入 PREPAR T, Y1, Y2 来通知 ACSL 模型在仿真时需要保留 T, Y1, Y2 三个参数，然后键入 START 来开始此模型的数字仿真。如果要得出系统的相平面图 (Y1-Y2)，则可以由 ACSL 的运行时命令 (run-time command) PLOT 来绘制所需的图形，亦即要想得出系统的相平面图，则可以键入如下命令：

```
PLOT Y1, Y2
```

这时在图形窗口上将显示出系统的相平面图如图 1-1 (a) 所示，这一曲线最终趋于封闭的图形。可见，在系统取了一个较远离原点的初始条件时，相平面图将逐渐收缩，最终将

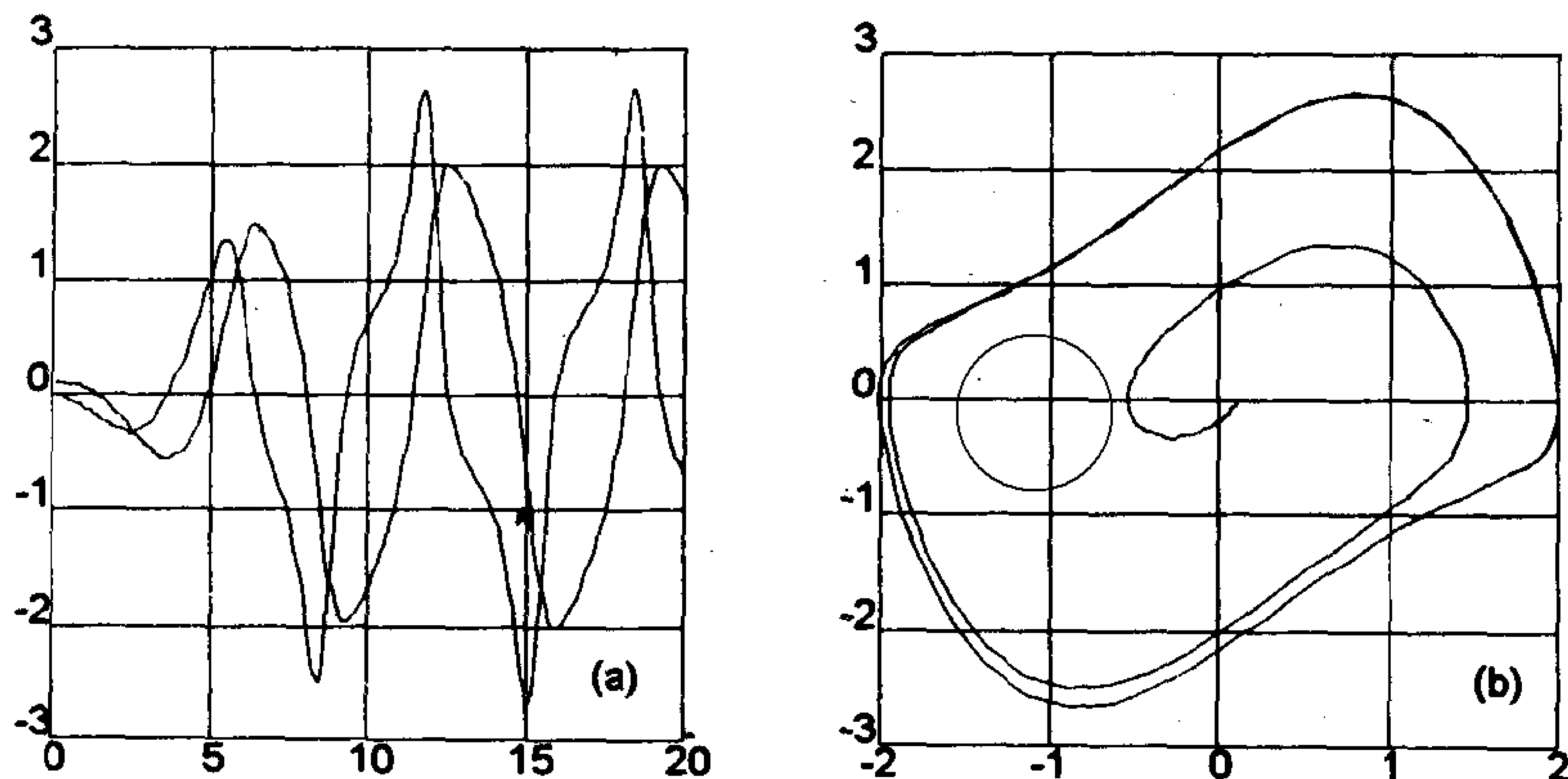


图 1-1　Van der Pol 方程的相平面图



稳定地收敛于这一条封闭的曲线。如果改变系统的初始条件，则可以利用 ACSL 的 SET 命令来实现：如 SET Y1C=0, Y2C=0.1, 这时再重复上述过程，则可以得出系统的相平面图如图 1-1 (b) 所示。可见，在系统初始条件取得较接近原点时，系统的相平面曲线将逐渐延伸，最后仍将收敛于一条封闭的曲线。由此可以定性地得出结论：无论系统的初始条件选作何值， Van der Pol 方程描述的系统最终的相平面曲线均将收敛于一条封闭的曲线，这一条曲线称为系统的极限环 (limit cycle) 。

ACSL 仿真语言近来又出现了一些新的进展，例如它提供了一个系统框图编辑界面，用户可以根据自己要研究的系统方框图直接“绘制出 ”ACSL 可以识别的框图形式，然后可以调用 ACSL 进行仿真，这样就显著地减少了用户使用该软件的难度，因为用户不必要去像以前那样记忆大量的 ACSL 函数，而直接可以通过可视的方法构成系统的模型来。

除了前面介绍的 ACSL 语言外还存在许许多多的其它仿真语言，如 ESL, TSIM, CSMP 等，例如在 ELS 语言下描述 Van der Pol 方程的语句和前面的类似，而得出的曲线如图 1-2 所示 [8] 。在这里是分别对 5 种 μ 值分别进行仿真的结果，其中每条曲线都由不同的标志给出。

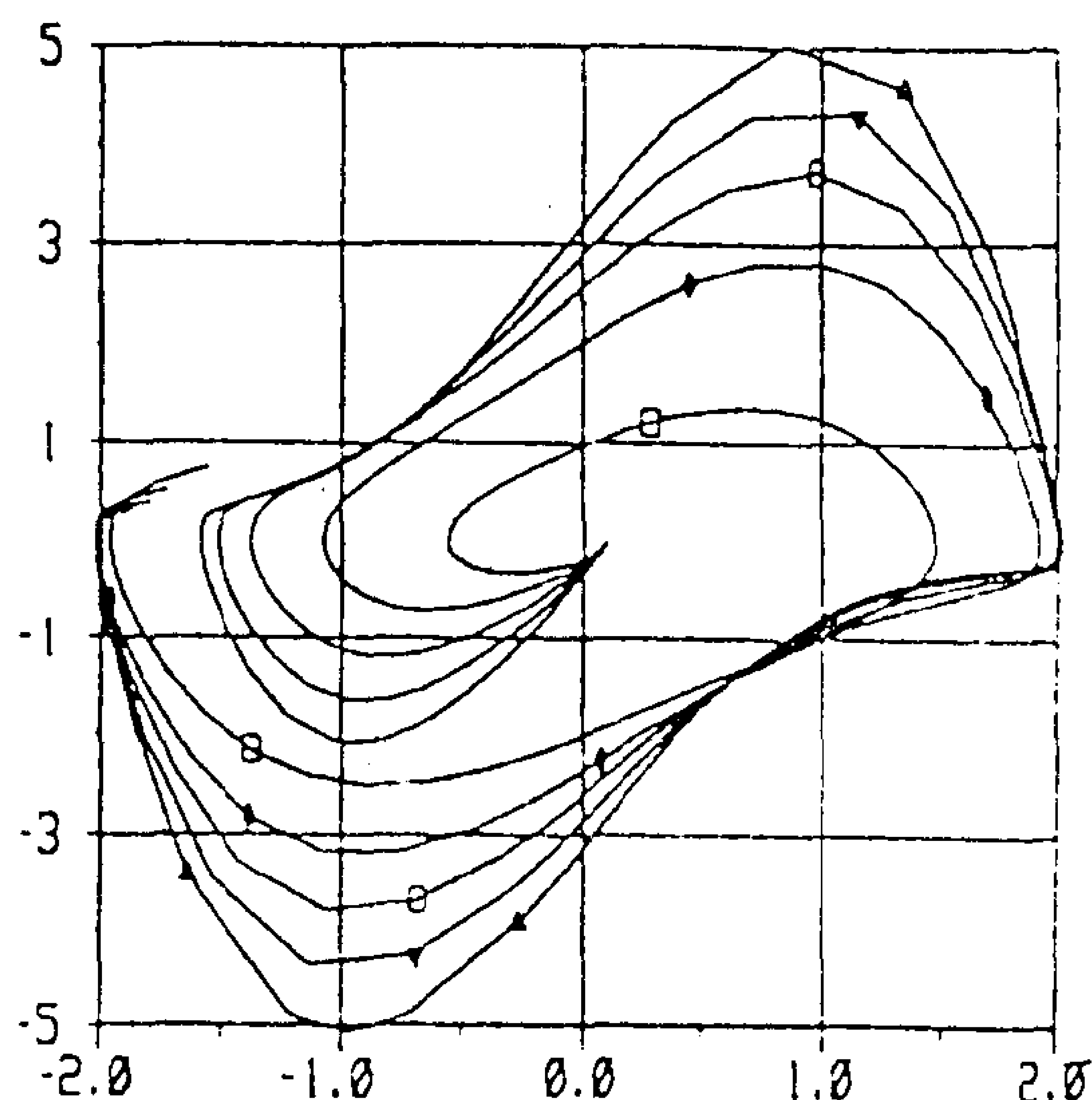


图 1-2　不同 μ 下 Van der Pol 方程相平面图

1.4　MATLAB, SIMULINK 与各种 CACSD 工具箱

1980 年美国的 Cleve Moler 博士研制的 MATLAB 环境 (或语言) 对后来的控制系统的理论及计算机辅助设计技术起到了巨大的推动作用，值得指出的是该语言原本并不是专门为控制理论领域的学者使用的，其最初目的是为线性代数等课程提供一种方便可行的实验手段，该软件出现以后一直在美国的 New Mexico 等大学作为教学辅助软件使用，并作为面向公众的免费软件 (public domain software) 广为流传，该软件的出现对控制界的影响是十分巨大的 [24] 。 MATLAB 于 1984 年推出了正式版本。

由于该软件的使用极其容易，且提供了丰富的矩阵处理功能，所以控制理论领域的研究人员很快就注意到了这样的特点，并在它的基础上开发了控制理论与 CAD 专门的应用程序集 (又称为工具箱)，使之很快地在国际控制界流行起来，目前它已经成为国际控制界最流行的语言了。除了流行于控制界以外， MATLAB 还在诸如图像信号处理、生物医学工程等相关的领域中有广泛的应用，并且出现了数以百计的各种实用工具箱，而这些工具箱反过来又进一步促进了 MATLAB 的应用。 MATLAB 当前的功能可以说是集可靠的数值运算 (特别是但不局限于矩阵运算) 、图像与图形显示及处理、高水平的



图形界面设计风格等于一身，此外它还提供了与其它高级程序设计语言 (对 MATLAB 来说是低级语言) 如 C, FORTRAN 等的接口，使得其功能日益强大，成为控制系统研究人员所不可缺少的有力工具。

由于 MATLAB 在控制界产生了巨大的吸引力，所以有很多软件设计人员仿照 MATLAB 的思想设计出了一些其它 CACSD 软件与环境。MATLAB 和其它交互式语言的出现标志着计算机辅助设计技术进入了一个崭新的阶段。

MATLAB 目前可以在下列各种类型的机器上运行：PC 及兼容机、Macintosh 、Sun 工作站、VAX 机、Apollo 工作站、HP 工作站、DECstation 工作站、SGI 工作站、RS/6000 工作站、Convex 工作站及 Cray 计算机等，可以说这些机器包括了一般常用的机器类型。如果单纯地使用 MATLAB 语言进行编程 (不采用其它外部语言)，则用 MATLAB 语言编写出来的程序不做丝毫地修改便可以直接移植到其它机型上去使用，所以说和其它语言不同，MATLAB 是和机器类型无关的，即使去设计像图形绘制或图形界面编辑这样往往依赖于机器类型的程序在 MATLAB 下也可以简单地将程序“拷贝”到其它机器下，而不必去担心它是否能在新的机器环境下正确运行。事实上，单纯地采用 MATLAB 语言可以实现 C 或 FORTRAN 等编程语言能做到的大部分工作，并且实现起来要比这些语言简洁方便得多。

依作者的观点，MATLAB 语言和 C 语言之间的关系与 C 语言和汇编语言之间的关系类似。例如在实际使用中，往往需要求出一个矩阵的特征值，而求取矩阵特征值的方法是多种多样的，如 QR 双步方法、Jacobi 方法等，所得出的结果不尽相同。但不管矩阵的参数如何，由 MATLAB 环境就可以由一条指令立即求出系统的特征值来，而不必去考虑是用什么算法以及如何实现等低级问题，也不必深入了解相应算法的具体内容。这就像在 C 语言下不必去深究乘法是怎样实现的，而只采用乘积的结果就可以了。

如果矩阵的条件数很大，则矩阵中的一个参数有极其微小的变化，将会使得最终结果发生极大的变化，这种现象在数学上称为坏条件问题。如果采用的方法不当，则最后得出的结果可能是不正确的，而采用 MATLAB 一般不会出现错误的结果，亦即 MATLAB 是可靠的、数值稳定的，而采用 C 或其它高级语言编写出来的程序可能得出错误的结果。

MATLAB 目前已经成为控制界国际上最流行的软件，它除了传统的交互式编程之外，还提出了丰富可靠的矩阵运算、图形绘制、数据处理、图像处理、方便的 Windows 编程等便利工具。此外，控制界很多学者将自己擅长的 CAD 方法用 MATLAB 加以实现，出现了大量的 MATLAB 配套工具箱，如控制界最流行的控制系统工具箱 (control systems toolbox), 系统辨识工具箱 (system identification toolbox), 鲁棒控制工具箱 (robust control toolbox), 多变量频域设计工具箱 (multivariable frequency design toolbox), μ 分析与校正工具箱 (μ-analysis and synthesis toolbox), 神经网络工具箱 (neural network toolbox), 最优化工具箱 (optimization toolbox), 信号处理工具箱 (signal processing toolbox) 以及仿真环境 SIMULINK 。参与编写这些工具箱的设计者包括国际控制界的名流，如 Alan Laub, Michael Sofanov, Leonard Ljung, Jan Maciejowski 等这些在相应领域的著名专家，这当然地提高了 MATLAB 的声誉与可信度，使得 MATLAB 风靡国际控制界，成为最重要



也是最流行的语言。

近来在国际上也出版了关于 MATLAB 及 CACSD 的专著和教材[16, 22, 25]，但它们大都是 MATLAB 的入门教材，并没有真正深入、系统地探讨 CACSD 技术及 MATLAB 实现。

1.5 符号运算系统 Mathematica 介绍

前面介绍的软件都是基于数值计算的，在实际研究中，除了数值计算之外，往往要求获得问题的解析解，这是 MATLAB 等软件所难以达到的。Mathematica 又有很强的绘图功能，所以在一些特殊的领域和应用中，符号运算工具如 Mathematica 的作用就显得很突出了。

Mathematica 是美国 Walfram Research 公司开发的产品，其研制者 Stephen Wolfram 在 1988 年前后正式推出了此软件。Mathematica 在符号运算软件中并不是较早出现的软件，早在它出现以前 Reduce, Maple 等软件已经较广泛和成功地被使用，然而 Mathematica 以其使用方便、功能强大、用户友好、扩展便利等特点很快就在符号运算类软件中占有统治地位，在数学研究的各个分支及其它领域都出现了 Mathematica 专用的扩展软件包[29]。

- Mathematica **的数值解法**: Mathematica 数值解的一个显著特点是可以按任意指定的精度(比如 100 位小数)去逼近原问题的解，而不是像 MATLAB 那样只能给出固定精度的解。

  利用 Mathematica 的数值运算功能可以容易地进行矩阵运算、方程求根、数值微积分，微分方程数值求解、最优化、特殊函数计算等常用数值问题，且可以按任意的精度得出问题的解，下面通过一个著名的例子演示 Mathematica 数值运算的精度。

例 1.1　第 3 章介绍 MATLAB 及代数方程求根的数值算法时将引出一个例子 $\prod_{i=1}^{20}(s+i)=0$ 的方程求根问题，显然方程的根为 $-1,-2,\cdots,-20$，经过其它数值算法之后，往往得不出较精确的解，这可以参照例 3.3 。然而使用 Mathematica 这类符号运算软件，可以容易地求出方程的根，在这里考虑一个更复杂的例子：求取 $\prod_{i=1}^{50}(s+i)=0$ 的根

```
In[1]:= Product[(x+i),{i,1,50}]
In[2]:=Solve[%==0,x]
Out[2]:={{x -> -50},{x -> -49},{x -> -48},{x -> -47},{x -> -46},{x -> -45},{x -> -44},
        {x -> -43},{x -> -42},{x -> -41},{x -> -40},{x -> -39},{x -> -38},{x -> -37},
        {x -> -36},{x -> -35},{x -> -34},{x -> -33},{x -> -32},{x -> -31},{x -> -30},
        {x -> -29},{x -> -28},{x -> -27},{x -> -26},{x -> -25},{x -> -24},{x -> -23},
        {x -> -22},{x -> -21},{x -> -20},{x -> -19},{x -> -18},{x -> -17},{x -> -16},
        {x -> -15},{x -> -14},{x -> -13},{x -> -12},{x -> -11},{x -> -10},{x -> -9},
        {x -> -8}, {x -> -7}, {x -> -6}, {x -> -5}, {x -> -4}, {x -> -3}, {x -> -2},
        {x -> -1}}
```




其中提示符 In[1] 引导的语句对给定的因式型多项式进行展开，而提示符 In[2] 引导的命令用来求解该高次方程。将第二条语句更换成 NSolve[%1==0,x,40] 也将获得同样的结果，这里的 40 字样表示保留 40 位小数的精度，可见利用 Mathematica 可以求出方程的精确解。由于这里所用到的所有数值位都全部保留，并没有任何近似，所以可以得出方程的解析解，另外的一个直接原因是在求解之前先试图进行因式分解运算，如果多项式可以精确地进行因式分解，则可以得出方程的根，而不是近似解，注意，这里给出的例子在 MATLAB 下会得出错误的结果。

- Mathematica **的解析解法**: Mathematica 的优势并不在数值运算上，其符号运算功能，亦即"解析解"求取方法是 MATLAB 这类数值软件所无法企及的。例如若已知线性系统的传递函数模型为

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

则可以通过下面的 Mathematica 命令求出系统的脉冲响应解析解

```
In[1]:= << Calculus`LaplaceTransform`
In[2]:=InverseLaplaceTransform[(s^3+7s^2+24s+24)/
       (s^4+10s^3+35s^2+50s+24),s,t]
Out[2]:= 4        6        2        -t
         ---- - ---- + ---- + E
          4 t      3 t      2 t
         E        E        E
In[3]:=TeXForm[%]
Out[3]:= ({4\over {{e^{4\,t}}}} - {6\over {{e^{3\,t}}}} +
         {2\over {{e^{2\,t}}}} + {e^{-t}})
```

这里首先调入 Laplace 变换软件包，然后由 InverseLaplaceTransform[] 命令求出系统的逆 Laplace 变换，此外还可以调用 TeXForm[] 函数将得出的结果用 TEX排版软件语句格式表示出来，这时可见系统的解析解为

$$\left(\frac{4}{e^{4t}}-\frac{6}{e^{3t}}+\frac{2}{e^{2t}}+e^{-t}\right)=4e^{-4t}-6e^{-3t}+2e^{-2t}+e^{-t}$$

还可以通过下面的例子领略 Mathematica 的解题方法和应用范围

```
In[1]:= Series[x^2 Sin[x]^2, {x,0,14}]
Out[1]:=        6        8       10        12         14
          4    x     2 x      x      2 x       2 x           15
         x  - -- + ---- - --- + ----- - ------ + O[x]
               3     45     315   14175    467775
In[2]:=Integrate[x^2 Sin[x]^2,x]
Out[2]:=    3                                         2
         4 x  - 6 x Cos[2 x] + 3 Sin[2 x] - 6 x  Sin[2 x]
         ---------------------------------------------------
                              24
```




```
In[3]:=m=Table[1/(i+j-1),{i,1,8},{j,1,8}]
In[4]:=Inverse[%]
Out[4]:={{64, -2016, 20160, -92400, 221760, -288288, 192192, -51480},
    {-2016,84672,-952560,4656960,-11642400,15567552,-10594584,2882880},
    {20160,-952560,11430720,-58212000,149688000,-204324120,141261120,-38918880},
    {-92400,4656960,-58212000,304920000,-800415000,1109908800,-776936160,216216000},
    {221760,-11642400,149688000,-800415000,2134440000,-2996753760,2118916800,- 594594000},
    {-288288,15567552,-204324120,1109908800,-2996753760,4249941696,-3030051024,856215360},
    {192192,-10594584,141261120,-776936160,2118916800,-3030051024,2175421248,-618377760},
    {-51480,2882880,-38918880,216216000,-594594000,856215360,-618377760,176679360}}
In[5]:=Det[m-x IdentityMatrix[8]]
Out[5]:=365356847125734485878112256000000-73867862762648220885539880960000 x +

                                               2                                   3
     2049469147096251407613473587200 x  - 5152730556096880878834155520000 x  +

                                           4                             5
     7393093004482064842152960000 x  - 3960868707237605474304000 x  +

                                 6           7     8
     507287185118380800 x  - 9052917760 x  + x
In[6]:=Det[m]
Out[6]:= 365356847125734485878112256000000
```

在前面例子中的前两个语句分别对 $f(x) = x^2 \sin^2 x$ 作前 14 项幂级数展开，并求出其不定积分，后面对一个 8 阶的 Hilbert 矩阵进行求逆、求特征多项式，并求出了该矩阵的行列式，注意这里得出的全部是解析解，并没有进行任何程度下的近似。其实 Mathematica 能做到的并不只是求出给定问题的解析解，它还可以应用于公式的推导与证明等高级的工作中。

- Mathematica **图形绘制功能**: Mathematica 还提供了较完善的图形绘制功能，例如若用户给出

  `Plot[Sin[1/x],{x,-1,1}]`

  命令，则将自动绘制出如图 1-3(a) 所示的二维图形，其中 {x, -1, 1} 表示自变量 x 在 $[-1,1]$ 区间取值。若用户给出

  `Plot3D[Exp[-(x^2+y^2)],{x,-2,2},{y,-2,2}]`

  命令则将绘制出如图 1-3(b) 所示的三维图形。

  Mathematica 还允许用户对得出的图形进行修饰，比如在图形上引入网格线、增高分辨率，还允许用户在图形上任意加注文本等修饰，总之用 Mathematica 可以创造出各种优美的图形修饰及输出效果。

- Mathematica **笔记本功能**: Mathematica 的笔记本是含有多级单元的交互式文件，每个单元中包括特殊的材料，这些材料可以是文本、图形或 Mathematica 的输入和输出，可以把这些材料有序地排放，形成自成体系的特殊说明材料。图 1-4 中给出了一个典型的 Mathematica 笔记本结构，用户可以充分利用 Mathematica 提供的笔记本功能来编写类似于菜单驱动的程序。



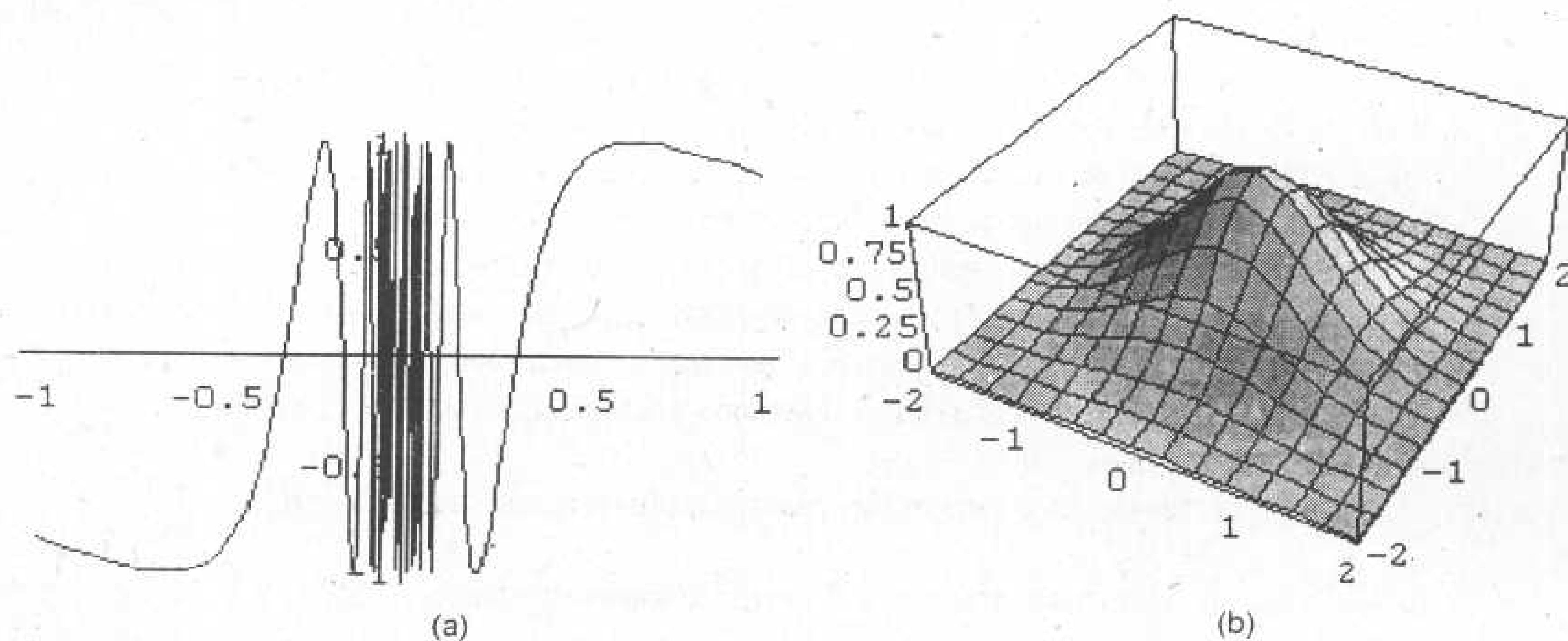


图 1-3　Mathematica 图形绘制效果

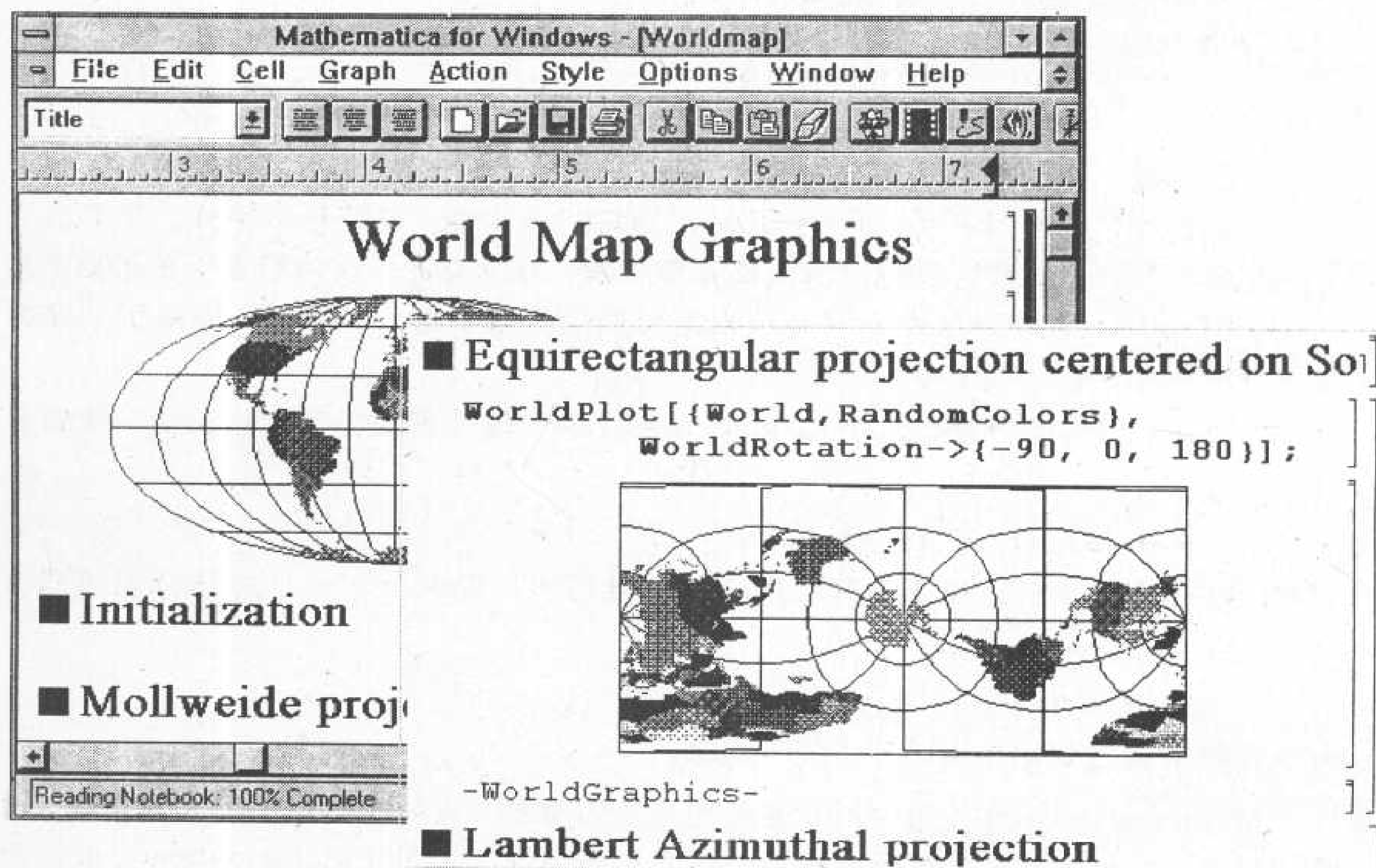


图 1-4　Mathematica 笔记本实例及效果

1.6　控制系统计算机辅助设计领域的新方法

在自动控制理论作为一门单独学科刚刚起步的时候，控制系统的设计是相当简单的，比如可以用 Ziegler-Nichols 经验公式利用纸和笔等简单的工具来设计较实用的 PID

控制器，这种现象持续了很长的时间。

随着计算机技术的发展，特别是像 MATLAB 这样方便可行的 CACSD 工具的出现，控制系统的计算机辅助设计在理论上也有了引人注目的进展，人们已经不满足于由纸和笔这样的简单工具设计出来的控制器了，而是期望越来越高。例如人们往往期望获得某种意义下的“最优”控制效果，而这样的控制效果确实是原来依赖纸和笔这样简单工具所实现不了的，而必须借助于计算机这样的高级工具，从而控制系统计算机辅助设计技术也就应运而生了。

早期的 CACSD 研究侧重于对控制系统的计算机辅助分析上，开始时人们利用计算机的强大功能把系统的频率响应图形绘制出来，并根据频率响应的曲线及自己的控制系统设计经验用试凑的方法设计一个控制器，然后利用仿真的方法去观察设计的效果，比较成功的试凑设计方法有超前滞后校正方法等，当然这样的方法更适合于单变量系统的设计，前面提及以 Rosenbrock 教授和 MacFarlane 教授为代表的英国学派多变量频率设计方法就是这种设计风格的范例 [1]。以色列 (美国) 学者 Issac Horowitz 教授在频域设计方法中独辟蹊径，创立了比较完善的设计方法 — 定量反馈理论，在反馈的效果上大作文章，在频域设计领域发展过程中，这些学者往往依赖于他们自己编写的 CACSD 工具来进行研究，并出现了很多值得提及的软件如 CLADP，后来随着 MATLAB 的发展，也出现了各种各样的 MATLAB 工具箱，如 Jan Maciejowski 等学者开发的多变量系统频域设计工具箱和美国学者 Craig Borghesani 和 Yossi Chait 等编写的 QFT 设计工具箱等。

除了经典的多变量频域方法之外，还出现了一些基于最优化技术的控制方法，其中比较著名的是英国学者 John Edmunds 提出的多变量参数最优化控制方法和英国学者 Zakian 提出的不等式控制方法 (method of inequalities) 等，这些方法都是行之有效的实用设计方法。

与此同时，美国学者似乎更习惯于状态空间的表示与设计方法 (往往又称为时域方法， time-domain)，首先在线性二次型指标下引入了最优控制的概念，并在用户的干预下 (如人工选择加权矩阵) 得出某种最优控制的效果，这样的控制又往往需要引入状态反馈或状态观测器等新的控制概念。此后为了考虑随机扰动的情况引入了 LQG 最优控制的设计方法，后来随着 LQG 控制固有的弊病提出了回路传输恢复 (loop transfer recovery, 简称 LTR) 等新技术，但直到这类状态空间方法找到了合适的频域解释之后才开始有了应用。此外在状态空间的设计方法中比较成型的方法有极点配置方法、多变量系统解耦控制设计等，这些状态空间方法在计算方法和理论证明上取得了很多的成果。

从控制系统的鲁棒性 (robustness) 角度也出现了各种各样的控制方法，首先由美国学者 Zames 提出的灵敏度最小控制策略引起了各国研究者的瞩目，并对之加以改进，出现了各种 H_∞ 最优控制的方案，所谓 H_∞ 实际上是物理可实现的稳定系统集合的一种数学描述 (因满足 Hardy 空间而得名)， H_∞ 控制的一个关键问题是 Youla 参数化方法，该方法可以给出所有满足要求的控制器的通式， H_∞ 的解法也是多种多样的，首先人们考虑通过 Youla 参数化方法构造出全部镇定控制器，并将原始问题转化成模型匹配

[1] 控制系统计算机辅助设计的方法在第 6, 7 章中还将在有关的地方给出综述，故在这里只给出简要的概念和发展情况，并不列出相应的原始文献，详细参考文献请参阅第 6,7 章。



(如 Hankel 近似) 的一般问题，然后再对该问题求解，后来多采用状态空间的解法，因为这样的解法更直观、容易，也更简洁。后来随着控制器的阶次越来越高，还出现了很多的控制器降阶方法来实现设计出的控制器。新近出现的线性矩阵不等式 (linear matrix inequalities) 及 μ 分析等控制系统设计方法也在控制界有较大的影响，而这些方法不通过计算机这样的现代化工具是不能完成的。

瑞典学者 Karl Åström 的研究似乎更加切合于过程控制的实际应用，在他的研究成果中经常可以发现独创性的内容，例如他和合作者对传统的，也是工业中应用最广泛的 PID 控制器进行了改进，提出了自整定 PID 控制器的思想，使得原来需要离线调节的 PID 控制器参数能够容易地在线自动调节，并在研究中取得了丰硕的成果，还推出了自整定 PID 控制器的硬件产品。在自整定 PID 控制器的领域也出现了很多比较显著的进展，这类研究的基本思想是使得复杂问题简单化，并易于实际应用。

1.7 本书的基本结构和内容

本书的第 1 章 (本章) 对国际上最流行的一些 CACSD 专用软件，如 ACSL, MATLAB, Mathematica 等作简要的介绍，然后对 CACSD 领域的新策略和新算法做一个概略的叙述，第 2 章将介绍 MATLAB 编程的基础，包括对赋值语句、控制结构和绘图语句等作较系统的介绍，第 3 章对数值线性代数的原理方法及 MATLAB 实现将做系统的叙述，并对其它数值分析问题，如数据处理、数值积分、非线性方程求解与最优化方法、以及常微分方程的数值解法及 MATLAB 实现将给出综合的介绍。第 4 章将对控制系统的数学模型及转换给出较详细的叙述，并将介绍模型的标准型求解、最小实现以及均衡实现等问题，然后还将对连续与离散时间系统的辨识与降阶问题作一个概略性的介绍。第 5 章将介绍控制系统仿真的基本方法，首先介绍单变量线性系统的频率响应，并给出线性系统的仿真方法，然后将通过 MATLAB 提供的仿真环境 SIMULINK 详细介绍非线性系统的仿真问题，最后还将介绍随机输入连续系统的仿真方法和非线性系统的频域分析方法。本书第 6 章和第 7 章将简要地介绍近年来发展起来的各种频域和时域 CACSD 的新方法与新策略，并配合 MATLAB 给出实用的求解方法及相关的工具箱介绍。这一部分的内容包括线性二次型 LQ 及 LQG 最优控制问题的求解方法、多变量系统频域设计方法、定量反馈理论 (QFT) 及其应用、PID 自整定控制策略、H_∞ 鲁棒控制及求解方法等，本章试图系统地介绍各种近年出现的控制策略，使读者更加开阔思路。第 8 章将介绍 MATLAB 的窗口环境图形用户界面的设计问题，并结合一个控制系统计算机辅助教学软件 Control Kit 来叙述 GUI 的具体实现方法。本书还给出了两个附录，其中附录 A 将列出 MATLAB 4.0 版的所有函数，而附录 B 将对各个函数作分类列表，以便读者参考及查询。

本书的前面几章详细而系统地介绍了 MATLAB 语言基本知识、编程方法和技巧，这当然可以直接适用于研究控制理论及控制系统计算机辅助设计技术的人员直接使用，此外同样适用于 MATLAB 语言的一般使用者。本书建议的阅读顺序如图 1-5 所示。



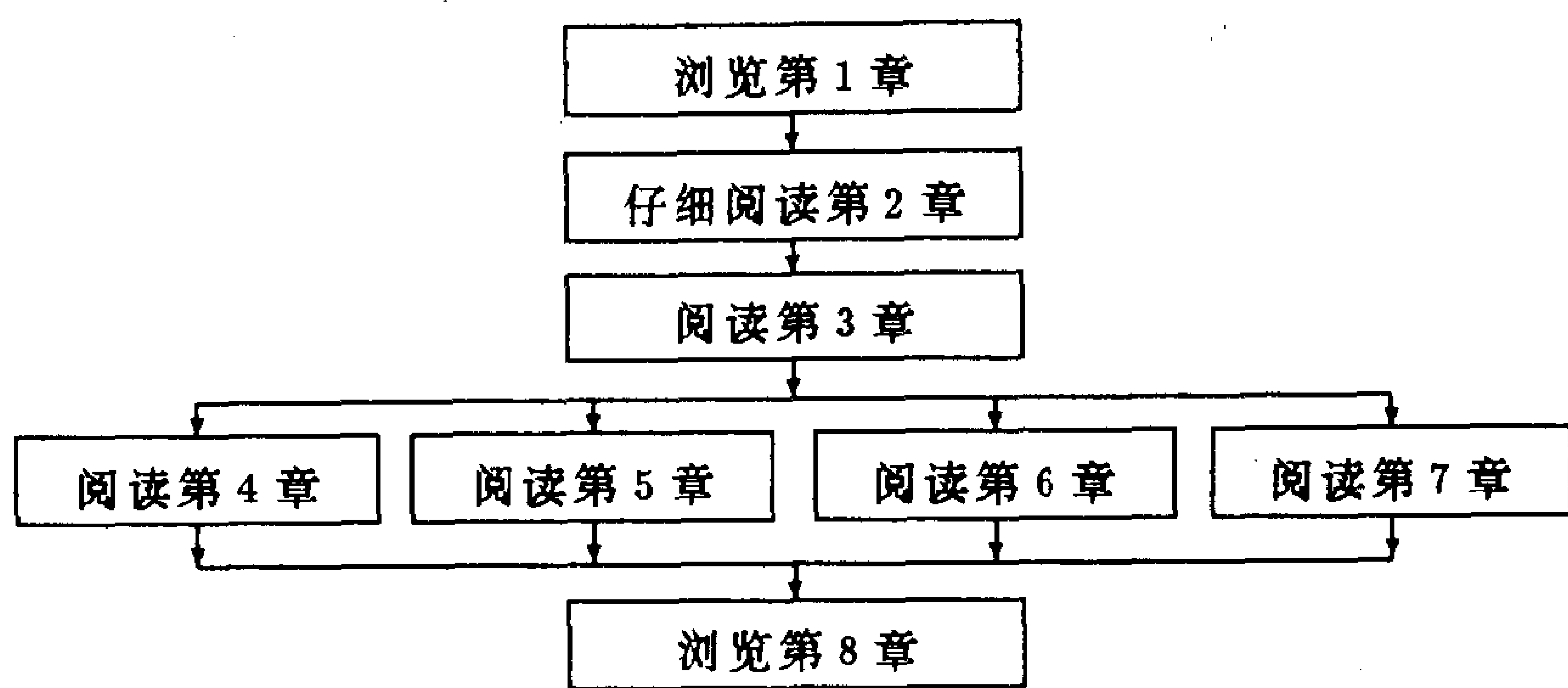


图1-5　本书的阅读顺序框图

参　考　文　献

[1] Armstrong E S. ORACLS – a design system for linear multivariable control. New York: Marcel Dekker Inc., 1980

[2] Åström K J. Computer aided tools for control system design. In:Jamshidi M, Herget C J. (eds.). Computer-aided control systems engineering. Amsterdam: Elsevier Science Publishers B V, 1985, 3-40.

[3] CAD Center. GINO-F Users' manual. 1976

[4] Dongarra J J, Bunch J R, Moler C B, Stewart G W. LINPACK user's guide, Philadelphia: Society of Industrial and Applied Mathematics (SIAM), 1979

[5] Edmunds J M. Cambridge linear analysis and design programs. Proceedings IFAC Symposium on CACSD. Zurich, Switzerland, 1979. 253-258

[6] Furuta K *et al.* Computer-aided design program for linear control systems", Proceedings IFAC Symposium on CACSD. Zurich, Switzerland, 1979. 267-272

[7] Garbow B S, Boyle J M, Dongarra J J, Moler C B. Matrix eigensystem routines – EISPACK guide extension. Lecture notes in computer sciences. Vol. 51. New York: Springer-Verlag, 1977

[8] Hay J L, Pearce J G, Turnbull L, Crosble R E. ESL software user manual. Salford: ISIM Simulation, 1988

[9] Herget C J, Laub A J (eds.). Special issue on computer-aided control system design. IEEE Control systems magazine, 1982, 2(4): 2-37

[10] Herget C J, Laub A J (eds.). Special issue on computer-aided control system design. Proceedings of IEEE, 1984, 72: 1714-1805

[11] Jamshidi M, Herget C J. (eds.). Computer-aided control systems engineering. Amsterdam: Elsevier Science Publishers B V, 1985



[12] Jamshidi M, Herget C J. (eds.) Recent advances in computer-aided control systems engineering. Amsterdam: Elsevier Science Publishers B V, 1992

[13] Jamshidi M S, Malek-Zavarel M. Linear control systems, computer-aided approach. Oxford: Pergamon Press, 1985

[14] Jobling C P, Grant P W, Barker H A, and Townsend P. Object-oriented programming in control system design: a survey. Automatica, 1994, 30: 1221-1261

[15] Kaga M, Furuta K. MATX: a high performance interactive software package for scientific and engineering computation. Proceedings IFAC Symposium on CACSD. Swansea. UK, 1991, 39-44

[16] Leonard N E, Levine W S. Using MATLAB to analyze and design control systems, 1993

[17] Maciejowski J M, MacFarlane A G J. CLADP: the Cambridge linear analysis and design programs. In: Jamshidi M, Herget C J. (eds.). Computer-aided control systems engineering. Amsterdam: Elsevier Science Publishers B V, 1985, 125-138

[18] Melsa J L, Jones S K. Computer programs for computational assistance in the study of linear control theory. New York: McGraw-Hill, 1973

[19] Mitchell and Gauthier Associate. Advanced continuous simulation language (ACSL) – user's manual, 1987

[20] Munro N. ECSTASY – a control system CAD environment. Proceedings IEE Conference on Control 88, Oxford, UK, 1988: 76-80

[21] Numerical Algorithm Group. NAG FORTRAN library manual. 1982

[22] Ogata K. Solving control engineering problems with MATLAB, 1994

[23] Press W H, Flannery B P, Teukolsky S A, and Vitterling W T. Numerical recipes, the art of scientific computing. Cambridge: Cambridge University Press, 1986

[24] Rimvall C M. Computer-aided control system design. IEEE Control Systems Magazine (Special issue on CACSD), 1993, 13: 14-16

[25] Shahian B, Hassul M. Computer-aided control system design using MATLAB. Englewood Cliffs: Prentice-Hall, 1993

[26] Smith B T, Boyle J M, Dongarra J J *et al.* Matrix eigensystem routines – EISPACK guide. Lecture notes in computer sciences. Vol. 6 (Second edition). New York: Springer-Verlag, 1976

[27] Spang III H A. The federated computer-aided design system. In: Jamshidi M, Herget C J. (eds.). Computer-aided control systems engineering. Amsterdam: Elsevier Science Publishers B V, 1985, 209-228

[28] 孙增圻，袁曾任．控制系统的计算机辅助设计，北京：清华大学出版社， 1988

[29] Wolfram Research Inc. Mathematica products catalog, 1995

[30] Wolfram S. Mathematica: a system for doing mathematics by computer (2nd edition). New York: Addison-Wesley, 1991



第 2 章 MATLAB 语言的使用与程序设计

2.1 MATLAB 简介

具有 FORTRAN 和 C 等高级计算机语言知识的读者可能已经注意到，如果用它们去进行程序设计，尤其当涉及矩阵运算或画图时，编程会很麻烦。比如说，若想求解一个线性代数方程，用户得首先去编写一个主程序，然后编写一个子程序去读入各个矩阵的元素，之后再编写一个子程序，求解相应的方程 (如使用 Gauss 消去法)，最后输出计算结果。如果选择的计算子程序不是很可靠，则所得的计算结果往往可能会出现问题。如果没有标准的子程序可以调用，则用户往往要将自己编好的子程序逐条地敲入计算机，然后进行调试，最后进行计算。这样一个简单的问题往往需要用户编写 100 条左右的源程序，键入与调试程序也是很费事的，并无法保证所敲入的程序 100% 地可靠，键入程序的过程中难免会出现误操作。求解线性方程组这样的一个简单的功能需要 100 条源程序，其它复杂的功能往往要求有更多条语句，如采用双步 QR 法求取矩阵特征值的子程序则需要 500 多条源程序，其中任何一条语句有毛病，甚至调用不当 (如数组维数不匹配) 都可能导致错误结果的出现。

MATLAB 的首创者 Cleve Moler 博士在数值分析，特别是在数值线性代数的领域中很有影响，他参与编写了数值分析领域一些著名的著作 [2, 3, 4, 5, 6, 9]。1980 年前后，Moler 博士在 New Mexico 大学讲授线性代数课程时，发现了用其它高级语言编程极为不便，便构思并开发了 MATLAB (MATrix LABoratory，即矩阵实验室)，这一软件利用了当时流行的 EISPACK [9] (基于特征值计算的软件包) 和 LINPACK [2](线性代数软件包) 两大软件包中可靠的子程序，用 FORTRAN 语言编写了集命令翻译、科学计算于一身的一套交互式软件系统。现在的 MATLAB 已经用 C 语言作了完全的改写。在 MATLAB 下，矩阵的运算变得异常的容易，后来的版本中又增添了丰富多彩的图形图像处理及多媒体功能。这一系统逐渐发展、完善，逐步走向成熟，形成了今天的模样。

由于 MATLAB 的应用范围越来越广，Moler 博士等一批数学家与软件专家组建了一个名为 MathWorks 的软件开发公司，专门扩展并改进 MATLAB。该公司于 1992 年推出了具有划时代意义的 MATLAB 4.0 版本，并于 1993 年推出了其微机版 [8]，可以配合 Microsoft Windows 一起使用，使之应用范围越来越广。1994 年推出的 4.2 版本扩充了 4.0 版本的功能，尤其在图形界面设计方面更提供了新的方法。

值得指出的是，MATLAB 一开始并不是为控制理论与系统的设计者们编写的，但 MATLAB 软件一出现很快就引起了控制界研究人员的瞩目，因为它把看起来那么繁琐的矩阵操作变得简单得令人难以置信，同时 MATLAB 还可以十分容易地绘制出各种精



美的图形。此外， MATLAB 又有那么好的可扩充性，可以把它当做一种更高级的语言那样去使用，用户可以用它容易地编写出各种通用或专用应用程序来。

由于 MATLAB 提供了强大的矩阵处理和绘图功能，很多控制界的名家在自己擅长的领域编写了一些具有特殊意义的 MATLAB 工具箱，如美国学者 Alan Laub 与 John Little 编写的控制系统工具箱 (control systems toolbox)，美国学者 John Little 和 Loren Shure 编写的信号处理工具箱 (signal processing toolbox), 瑞典学者 Leonard Ljung 的系统辨识工具箱 (system identification toolbox)，美国籍学者 Richard Chiang 与 Michael Sofanov 的鲁棒控制工具箱 (robust control toolbox), μ 分析与综合工具箱 (μ-analysis and synthesis toolbox), 美国学者 Craig Borghesani, Yossi Chait 和 Oded Yaniv 设计的定量反馈理论工具箱 (QFT toolbox), 美国学者 Howard Demuth 与 Mark Beale 的神经网络工具箱 (neural network toolbox), 英国学者 Jan Maciejowski 等的多变量频域设计工具箱 (multivariable frequency design toolbox)，英国学者 Peter Fleming 及美国学者 Andrew Grace 编写的两个版本的最优化工具箱 (optimisation toolbox) 等等，1992 年 MathWorks 公司推出的交互式模型输入与仿真环境 SIMULINK 更使得 MATLAB 为控制系统的仿真与 CAD 中的应用打开了崭新的局面。

目前， MATLAB 已经成为国际上最流行的控制系统计算机辅助设计的软件工具，现在的 MATLAB 已经不仅仅是一个“矩阵实验室”了，它已经成为了一种具有广泛应用前景的全新的计算机高级编程语言了。

MATLAB 是以复数矩阵作为基本编程单元的一种程序设计语言，它提供了各种矩阵的运算与操作，并有较强的绘图功能，所以得以广为流传，成为当今国际控制界应用最广，也是最受人们喜爱的一种软件环境。MATLAB 是一个高度的集成系统， MATLAB 4.0 版本更是集科学计算、图像处理、声音处理于一身。另外它提供了丰富的 Windows 图形界面设计方法，为用户在不失强大功能的前提下设计出友好的图形界面提供了便利的工具。MATLAB 不仅流行于控制界，在生物医学工程、语音处理、图像信号处理、雷达工程、信号分析、计算机技术等各行各业中都有极广泛的应用。

MATLAB 和其它高级语言之间的关系仿佛该高级语言和汇编语言的关系一样，因为高级语言的执行效率要低于汇编语言，然而其编程效率与可读性、可移植性要远远高于汇编语言。同样 MATLAB 比一般高级语言的执行效率要低，而其编程效率与可读性、可移植性要远远高于其它高级语言，所以在计算机辅助设计与仿真中较适合于从像 MATLAB 这样的专用高级语言入手，这不但可以大大地提高编程的效率，而且可以大大地提高编程的质量与可靠性。其实对于专门从事 CACSD 研究的人员来说，因为 MATLAB 语言可以轻易地再现 C 或 FORTRAN 语言几乎全部的功能，所以即使用户不懂 C 或 FORTRAN 这样的程序设计语言也照样可以设计出功能强大、界面优美、稳定可靠的高质量程序来，且开发周期会大大地缩短。

MATLAB 在各种常用的计算机系统如 IBM 兼容微型机、SUN 工作站、VAX 系统机、苹果 Macintosh 微型机、 Apollo 工作站和其它一些机器上完全兼容，即使在一些特殊的文件形式 (如二进制码) 有所不同时， MATLAB 还允许对之进行传递和转换。就 IBM 兼容机而言， MATLAB 既可以在 MS-DOS 下运行，也可以在 MS-Windows 下运



行。此外还可以在一些其它环境如 X-Windows 下运行，所以一般说来， MATLAB 并不受机器及环境的限制。 MATLAB 本身提供了较完善的联机 (on line) 帮助软件环境，用户可以通过它查询一些不熟悉的命令的功能与调用格式，还提供了程序演示的功能，MATLAB 4.0 版本又提供了图形界面的开发功能，适用于 Microsoft Windows 等应用窗口的编程。

严格地说， MATLAB 并不是一种计算机语言，因为用它编写出来的程序并不能脱离 MATLAB 环境而执行，但从其功能上说， MATLAB 已经完全具备了计算机语言的结构与性能，所以本书中将称 MATLAB 为语言。

2.2 MATLAB 环境的安装与基本操作

目前比较流行的 MATLAB 版本是 MATLAB 4.0 ，本书中所有的叙述都是基于 MATLAB 4.x 的，同时也适当地兼顾 3.5 版本，其中的一些功能和语句在 MATLAB 3.5 上不一定能实现。关于各个 MATLAB 版本之间的区别，用户可以自己去查阅有关手册，这里就不再赘述了。在附录 A 中给出了标准 MATLAB 的函数名简表，用户可以通过这一简表查询有关的 MATLAB 函数与命令名，然后再根据联机帮助系统来获得所需要的信息。

在 PC 兼容机下使用 MATLAB 的前提条件是该机器配备有协处理器芯片 (math-coprocessor, 即 287 或 387)，当然 486DX 以上的机型因为协处理器在 CPU 上已经存在，所以不必另配置该芯片。运行 MATLAB 4.x 版本的硬件条件更为苛刻，它要求首先运行 386 增强型的 Microsoft Windows 环境，且有 4MB 以上的内存，要实现其中一些特殊的功能则需要有 8MB 的内存，当然这些条件在计算机技术飞速发展的今天已经不能成为阻碍 MATLAB 广泛应用的因素了。

若想安装 MATLAB 4.x 版，首先应该启动 Microsoft Windows 环境，再将第一片盘插入 A 驱动器，然后在程序管理器下选择 File 菜单下的 Run 菜单项，从编辑框中键入 a:setup ，然后按下 OK 按钮即可，安装程序就会提示用户将各个盘上的内容复制到硬盘上。 Windows 版的 MATLAB 安装起来之后，则将获得如图 2-1 所示的组窗口，其中标注为 MATLAB 4.0 with Simulink 的图标对应于 MATLAB 的执行程序，用户可以用鼠标左键双点相应图标的方法来启动该程序，这时将出现如图 2-2 所示的界面显示，在该界面下的 >> 标志为 MATLAB 的命令提示符，用户可以在 >> 提示符下输入 MATLAB 命令。如果用户是第一次使用 MATLAB ，则建议首先在 >> 提示符下键入 DEMO 命令，它将启动 MATLAB 的演示程序，用户可以在此演示程序中领略 MATLAB 所提供的强大的运算和绘图功能。

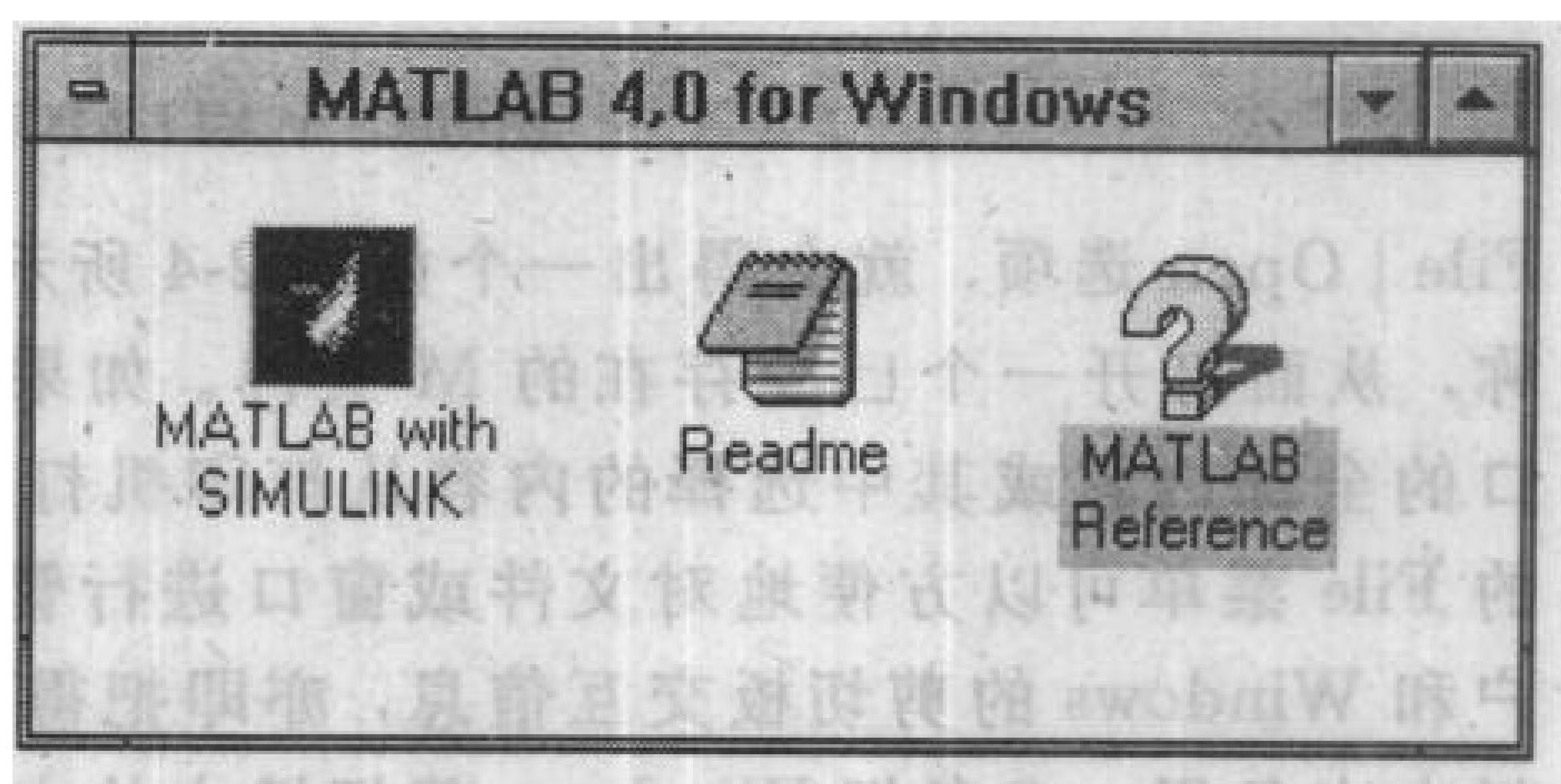


图 2-1 MATLAB 程序组窗口内容

MATLAB 4.0 版是一个高度集成的语言环境，在它的界面下可以编写程序、运行程

序并跟踪调试程序。从图 2-2 可以看出，MATLAB 命令窗口的界面下有一个菜单条，其中提供了很多有用的功能，如其中的 File (文件管理) 菜单和 Options (选项) 菜单下

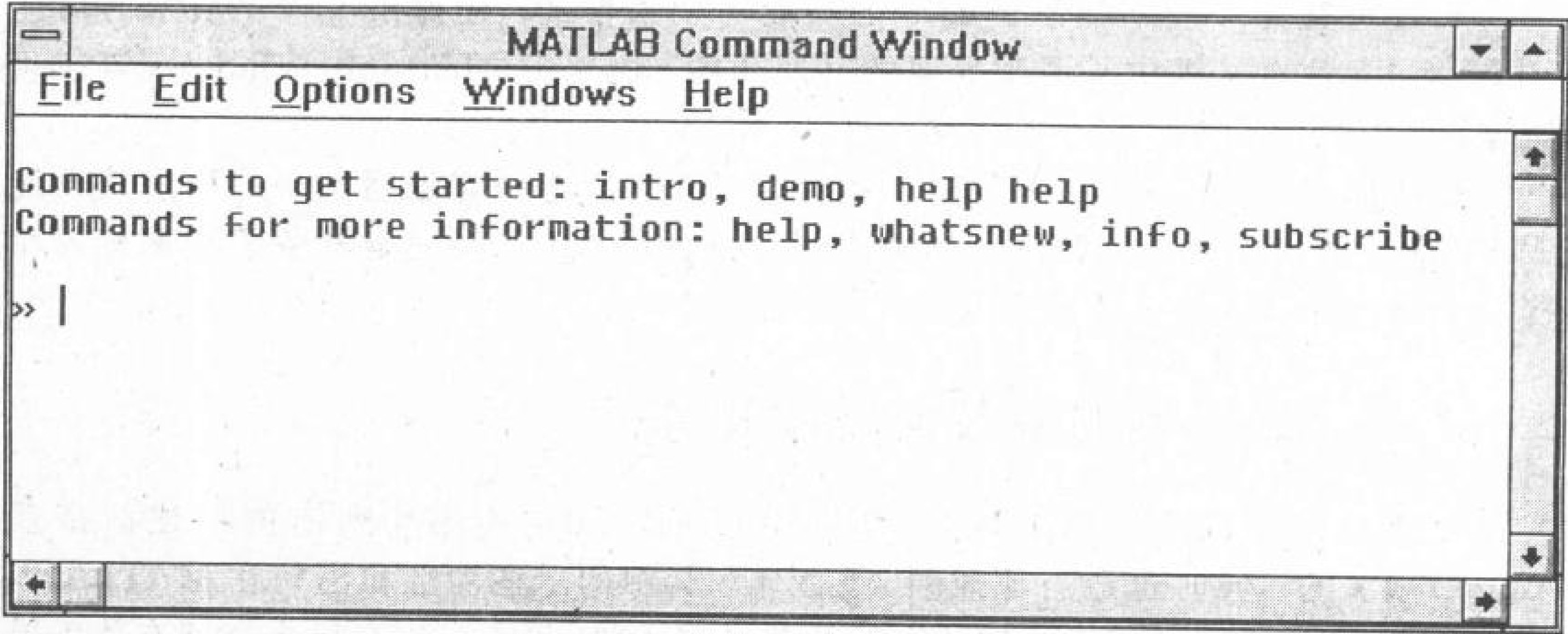


图2-2 MATLAB 4.0 程序界面

都有很多子菜单项，这两个菜单条的内容如图 2-3(a) 和 (b) 所示。选择了 File | New 选项之后，就会得出一个如图 (a) 所示的子菜单，它允许用户打开一个新的文件或模块，如 M-File (M 文件)、Figure (图形窗口) 或 Model (SIMULINK 编辑界面)。如果选择了

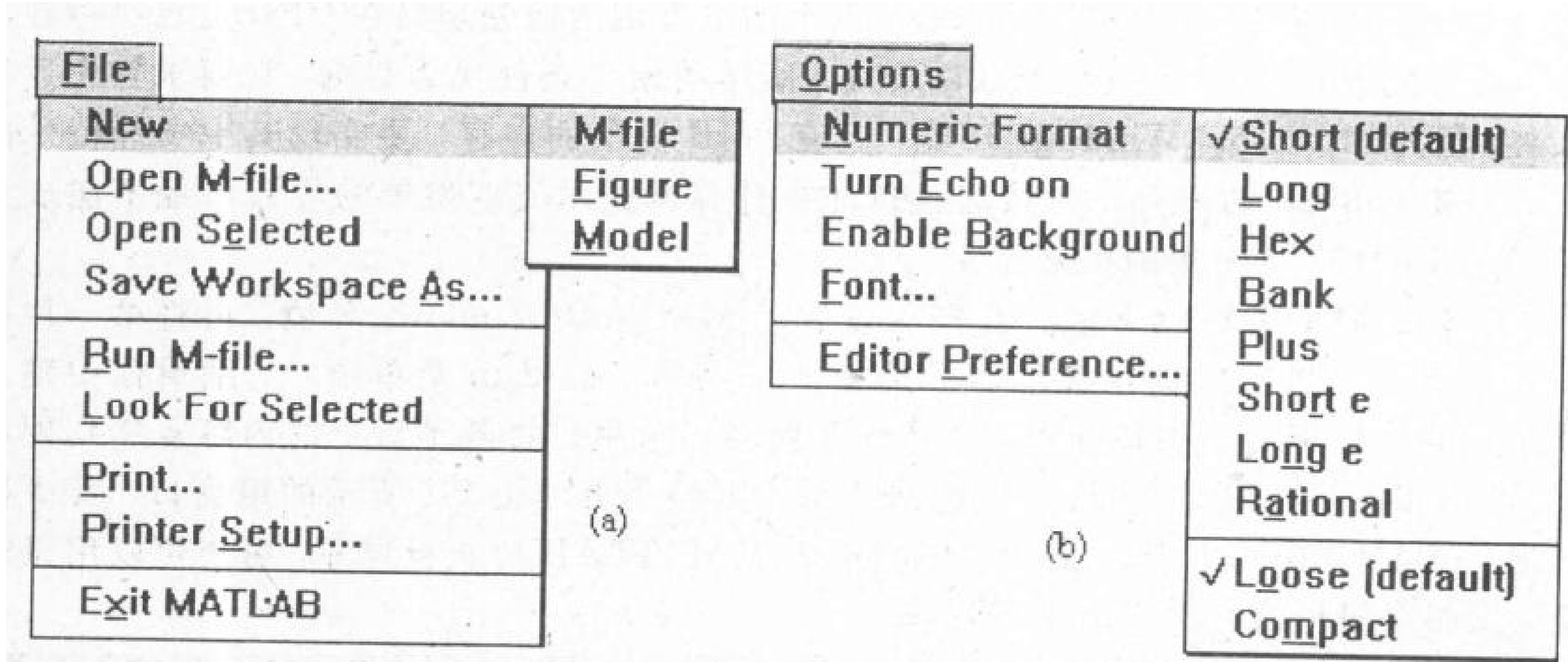


图2-3 MATLAB 程序的常用菜单

File | Open 选项，就会得出一个如图 2-4 所示的对话框，用户可以从中选择 M 文件的名称，从而打开一个已经存在的 M 文件。如果选择了 File | Print 选项，则可以将命令窗口的全部内容或其中选择的内容从打印机打印出来。总之，利用 MATLAB 界面下提供的 File 菜单可以方便地对文件或窗口进行管理。MATLAB 的 Edit (编辑) 菜单允许用户和 Windows 的剪切板交互信息，亦即把得出的结果由 Edit | Copy 菜单项复制到剪切板上去备用，或者把 Windows 剪切板中的内容由 Edit | Paste 菜单项复制到 MATLAB 的命令窗口中来。还可以调用 Edit | Clear Session (清除一次运行的中间结果)，这相当于在 MATLAB 环境下给出 `clear` 命令。

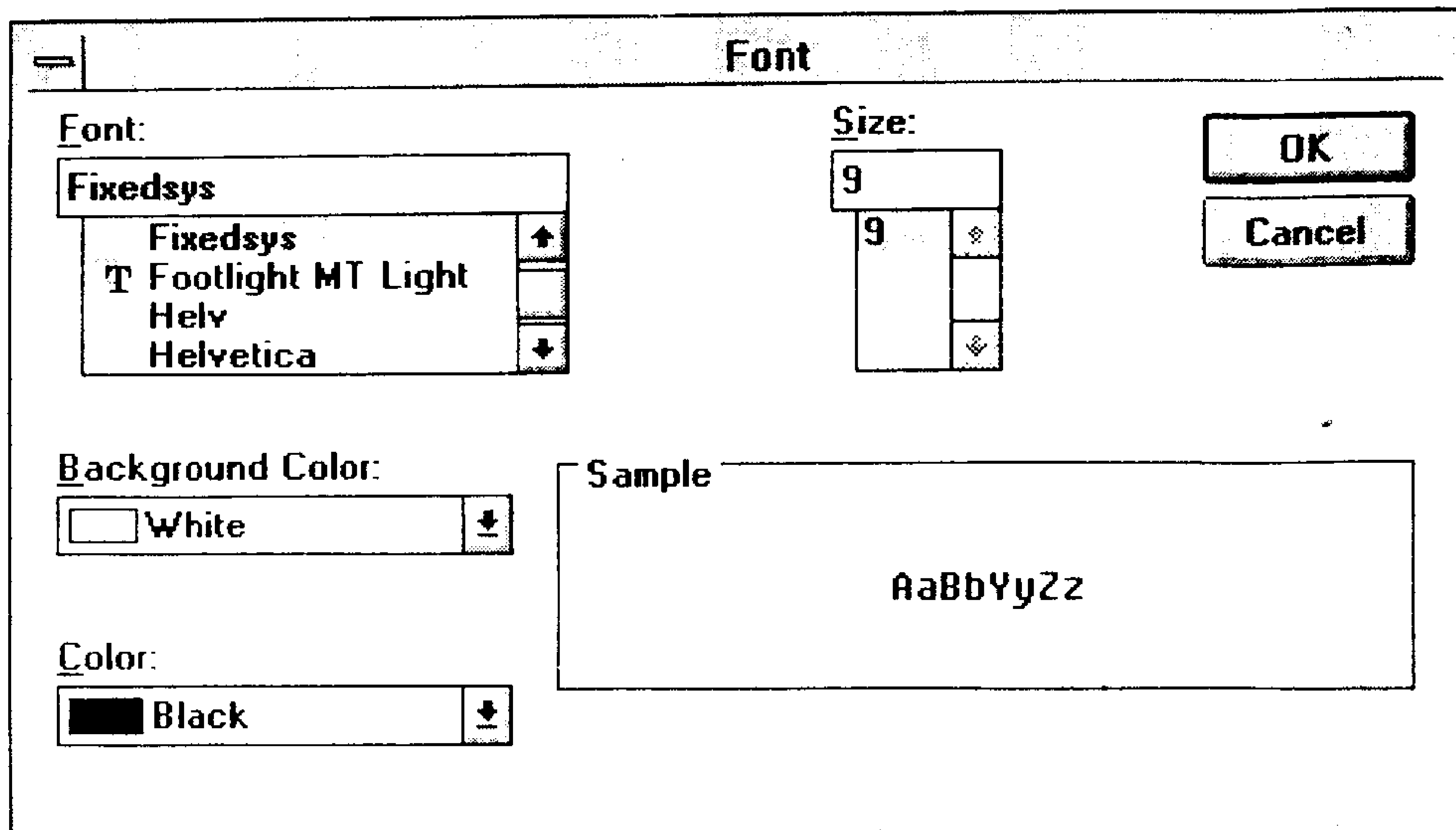


图2-4 打开文件对话框

如果用户选择了 Options | Numeric Format (数字显示格式) 就可以得出如图 2-3(b) 所示的下一级子菜单，允许用户设置 MATLAB 下结果数据的显示格式，用户可以从中选择 Short (默认的简洁格式) 和 Long (高精度格式) 等，通过这个选项的设置，用户就可以得到期望的显示格式了。各种显示格式的设定同样可以通过 MATLAB 命令 format 来完成，这在后面还要介绍。Options | Turn Echo on/off (开通或关断命令显示) 可以设置是否显示各条 MATLAB 命令，它的作用相当于 MATLAB 的 echo on 或 echo off 。Enable/Disable Background Process 将指定是否允许背景处理。Options | Font (字体设置) 菜单项将给出一个如图 2-5 所示的对话框，允许用户设置命令窗口内的显示字体类型及前景背景颜色等。

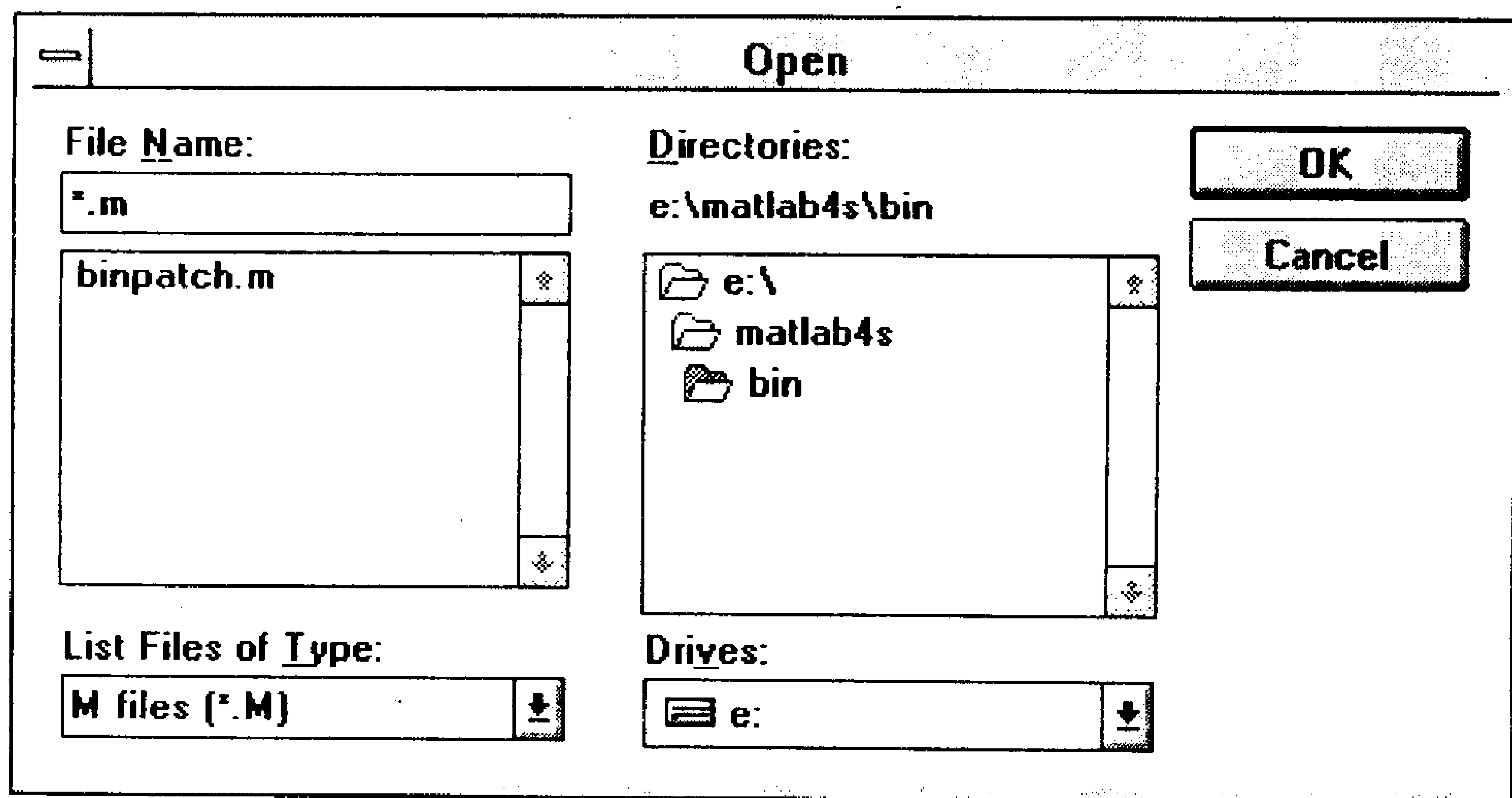


图2-5 字体设置对话框



MATLAB 界面下 Windows(窗口管理) 选项允许用户在各个 MATLAB 窗口中作直接的切换，如果同时打开了若干个不同的窗口，则可以给出所有打开窗口名的列表，用户可以直接从列表中选择想调到前台的窗口。

运行完 MATLAB 之后，就可以在提示符下键入 exit 或 quit 命令来退出 MATLAB 环境，并回到 Microsoft Windows 的界面下。当然也可以由选中 File | Exit MATLAB 菜单项来退出 MATLAB 环境。

2.3 MATLAB 的基本语句结构

MATLAB 实际上可以认为是一种解释性语言，用户可以在 MATLAB 工作环境下键入一个命令，也可以由它定义的语言编写一个或多个应用程序，这样 MATLAB 软件对此命令或程序中各条命令进行翻译，然后在 MATLAB 环境下对它进行处理，最后返回运算结果。MATLAB 以复数矩阵为最基本的运算单元，既可以对它整体地进行处理，也可以对它的某个或某些元素进行单独地处理，所以操作起来比较方便。MATLAB 语言最基本的赋值语句结构为

```
变量名列表 = 表达式
```

其中等号左边的变量名列表为 MATLAB 语句的返回值，等号右边的是表达式的定义，它可以是 MATLAB 允许的矩阵运算、也可以包含 MATLAB 下的函数调用。

- 等号右边的表达式可以由分号结束，也可以由逗号或换行号结束，但它们的含义是不同的。如果用分号结束，则左边的变量结果将不在屏幕上显示出来，否则将把左边返回矩阵的内容全部显示出来。
- 和 C 语言类似，MATLAB 是区分变量名的大小写的 (case-sensitive)，例如在 MATLAB 下可以用 Abc 、 ABC 和 abc 来表示不同的矩阵名，但在实际编程时还应慎重，尽量避免这样的变量命名方式。
- MATLAB 和 C 语言不同，在调用函数时 MATLAB 允许一次返回多个结果 (亦即多个矩阵)，这时等号左边是由 [ ] 括起来的矩阵列表，例如

```
[m, p] = bode(n, d, w)
```

中调用了控制系统工具箱中的 bode() 函数来求取传递函数 n，d 在指定的频率段 w 内的幅值响应 m 与相位响应 p，可见调用了这一函数之后，就可以在等号左边同时返回两个矩阵 m 和 p，有关 bode() 函数的内容及调用请参见第 5 章。

- 从前面的例子可以看出，MATLAB 函数调用时输入和输出变量分别在等号的两端列出，这种记号很容易理解。此外，在调用上述函数时还可以采用下面的格式

```
[m, p] = bode(a, b, c, d, 1, w)
```




其中 a, b, c, d 为系统的状态方程描述, MATLAB 会自动地从输入参数的个数上判定给出的是传递函数还是状态方程模型, 从而进行正确的计算, 这类似于 C++ 中重载的概念。可见 MATLAB 及各个工具箱为用户提供了极大的方便。

在 MATLAB 环境下, 矩阵输入的方式是很直观的, 如矩阵 $A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$ 的输入方式为 A=[1,2,3; 4,5,6; 7,8,0], 其中同一行中的内容用逗号分隔, 而采用分号来表示换行。按这种格式输入矩阵 A 后, 矩阵的内容将在屏幕上按照下面的格式显示出来 [1)]

```
>>  A=[1, 2, 3; 4, 5, 6; 7, 8, 0]
A =  1    2    3
     4    5    6
     7    8    0
```

如果在上面赋值的式子的末尾加一个分号, 则矩阵的内容就不在屏幕上显示了。所以用户可以通过是否在语句末加分号的方式来决定运算的结果是否显示出来, 这样就可以使得不必要的中间结果部分不被显示出来。

在一般情况下, 用于同行中分隔的逗号是可以由空格来代替的。其实, MATLAB 的矩阵输入格式并不是很严格, 在语句中多加入一些空格将不会影响整个赋值结果。此外, 前面的矩阵还可以等价地由下面两种方式来输入

```
>>  A=[1 2 3; 4 5 6                >>  A=[1 2 3; 4,5,...
       7 8 9];                              6; 7 8, 9];
```

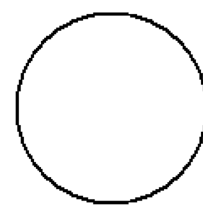
其中, 在前一种输入方法中, 第二行的末尾用一个回车键来表示换行, 它的作用和一个分号是一致的。如果用户采用后面一种输入方式, 则第一行末尾给出的三个点号 (···) 称为续行符号, 它表示下面的一行应该紧接在前一行上。在一个语句相当长, 难以由一行表示出来时, 往往需要采用这样的续行符号, 将一条语句分隔成几行来表示。

向量的输入是一般矩阵输入的特殊情况, 例如下面的两条命令

```
>> V1=[1 2 3,4],          >> V2=[1; 2; 3; 4]
```

可以简单地输入下面给出的行向量和列向量

```
V1 =   1    2   3    4          V2=  1
                                     2
                                     3
                                     4
```



除了表示向量和矩阵之外, 利用 MATLAB 当然可以容易地表示标量了。学会了矩阵的基本表示方法之后, 就可以容易地理解下面的输入方式和结果了

```
>>  A=[A; [1 3 5]]
A = 1    2    3
```

[1)] 由于不能无谓地占用过多的篇幅, 所以这里及书中后面的部分只列出显示内容中有用的部分, 而一些空行将被略去。



```
     4    5    6
     7    8    0
     1    3    5
```

可见，这种方法的作用是在原来 A 矩阵的下面再附加上一个行向量 [1 3 5]。如果矩阵各行的元素个数不相同时，例如写成 A=[A; [1 2]], 这样将出现下面的错误信息

```
>> A=[A; [1,2]]
??? All rows in the bracketed expression must have the same
number of columns.
```

提示该矩阵有误 (行中元素个数不匹配)，并给出一声鸣叫，从而提示用户重新输入矩阵。

MATLAB 定义了两个基本的复数常量， i 和 j, 这些值为数学上的 $\sqrt{-1}$。如果它们不被重新赋值，则将保留这种定义。如果重新赋值，则这两个变量将保留新的值。如果想把一个变量 a 赋成 $\sqrt{-1}$，则采用 MATLAB 命令 a=sqrt(-1) 即可。

在 MATLAB 下 4+3*i 的赋值命令将得出 4.0000 + 3.0000i 的结果，而利用前面的 A 矩阵，在输入命令 A+ones(size(A))*i 后可以得出下面的结果

```
>> A+ones(size(A))*i
ans =  1.0000 + 1.0000i   2.0000 + 1.0000i   3.0000 + 1.0000i
       4.0000 + 1.0000i   5.0000 + 1.0000i   6.0000 + 1.0000i
       7.0000 + 1.0000i   8.0000 + 1.0000i   0.0000 + 1.0000i
       1.0000 + 1.0000i   3.0000 + 1.0000i   5.0000 + 1.0000i
```

其中 ones(size(A)) 命令将产生一个和 A 矩阵同样大小的、元素全部为 1 的矩阵。在书写带有复数项的矩阵时应该注意，不能在一个矩阵元素之内再加空格，否则将得出错误的结果。例如 [1 +2*i, 2] 赋值表达式因为在 1 和 +2*i 之间无意地多留了一个空格，所以将被错误地解释成含有三个元素的向量。

一般说来，在 MATLAB 环境下矩阵名可以为任意字符串，但是 MATLAB 保留了一些特殊的字符串，如判断 0 元素用的误差限 eps, 其默认值为 eps=2.2204×10^{-16}， pi 表示圆周率 π 的值，Inf 表示无穷大 ∞，MATLAB 允许的最大数据为 1.7976931×10^{308}，一个数据大于此数则认为是 Inf 。Inf 的另一种产生方法为计算 1/0 的值，这种运算在一般计算机语言中会被认为是“非法除 0”而将中止整个程序的运行，而 MATLAB 依照 IEEE 标准允许这种运算，不会因此产生异常中止，只给出 “Warning: Divide by zero” 这样的警告信息。例如 MATLAB 允许作这样的矩阵赋值 A=[1 2 Inf;1 2 5]，其结果将产生下面的矩阵

```
>> A=[1 2 Inf; 1 2 5]
A =  1    2    Inf
     1    2    5
```

而不会影响到其它的矩阵元素。MATLAB 还可以求出该矩阵与列向量 [1;2;3] 的乘积为



```
>>  A*B
ans =  Inf
        20
```

这在其它语言下是不可能的。 MATLAB 中保留的 NaN 常量，它是一个不定式 (Not a Number)，是由 Inf/Inf 或 0/0 这样的运算产生的。从而可见， MATLAB 有着比其它语言更高的容错性，使得它更为可靠。

在变量赋值语句中，等号左边的矩阵名列表和等号一起可以省略，这时将把返回的矩阵名设置为 ans 变量。 ans 也是一个保留的 MATLAB 字符串，它表示上面一个式子的返回结果。例如如果键入 [1 2, 3; 4, 5 6] 将产生下面的输出

```
>>  [1 2, 3; 4, 5 6]
ans =  1   2   3
       4   5   6
```

这时 MATLAB 会自动地将该矩阵赋给 ans 变量，这样用户就可以对 ans 变量进行直接操作了。即使在 MATLAB 中保留了若干字符串，它们还可以重新进行再赋值，例如如果用户想将判 0 用的误差限扩大十倍，则可以采用 eps=10*eps 命令来进行修正，这时如果再使用 eps 变量时则会自动取修正后的数值了。重新启动 MATLAB，则被改变的保留常量将恢复到原来的数值。应该注意，如果对 Inf 常量重新赋值，比如赋值为 Inf=10，则该变量将失去了原来的无穷大意义。

2.4 矩阵的基本运算

如果一个矩阵 A 有 n 行、m 列元素，则称 A 矩阵为 $n\times m$ 矩阵，如果 $n=m$，则矩阵 A 又称为方阵。 MATLAB 定义了下面各种矩阵的基本运算：

- **矩阵加减法运算**：假设在 MATLAB 工作环境下有两个矩阵 A 和 B，则可以由下面的命令执行矩阵加减法： C=A+B 和 C=A-B 。若 A 和 B 矩阵的维数相同，则可以执行矩阵的加减法，它会自动地使得 A 和 B 矩阵的相应元素相加减。如果 A 与 B 的维数不匹配，则 MATLAB 将自动地给出错误信息，提示用户两个矩阵的维数不匹配。
- **矩阵的转置**: 在数学公式中一般把一个矩阵的转置记作 A^T，假设 A 矩阵为一个 $n\times m$ 矩阵，则其转置矩阵 B 的元素定义如下 $b_{ji}=a_{ij}, i=1,\cdots,n, j=1,\cdots,m$ 。例如，从下面给出的 A 矩阵可以容易地求出其转置矩阵 A^T：

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix},\quad A^T=\begin{bmatrix}1&4&7\\2&5&8\\3&6&0\end{bmatrix}$$

若 A 矩阵含有复数元素，则对之进行转置时，其转置矩阵 B 的元素定义如下 $b_{ji}=a^*_{ij}, i=1,\cdots,n, j=1,\cdots,m$，亦即首先对各个元素进行转置，然后再逐项求取



其共轭复数值，这种转置方式又称为 Hermit 转置，其数学记号为 $B = A^*$ 。例如下面给定复数矩阵及 Hermit 转置分别为

$$A = \begin{bmatrix} 5+i & 2-i & 1 \\ 6i & 4 & 9-i \end{bmatrix}, \quad B = A^* = \begin{bmatrix} 5-i & -6i \\ 2+i & 4 \\ 1 & 9+i \end{bmatrix}$$

在 MATLAB 下，矩阵 A 的转置 (包括复数矩阵的 Hermit 转置) 可以简单地由 A' 求出，例如上面转置可以由以下语句来实现

```
>>  A = [5+i, 2-i, 1; 6*i, 4, 9-i]; B=A'
ans =   5.0000 - 1.0000i        0 - 6.0000i
        2.0000 + 1.0000i   4.0000
        1.0000             9.0000 + 1.0000i
```

- **矩阵的翻转处理**: MATLAB 还提供了一些矩阵翻转处理的特殊命令，如 B=fliplr(A) 命令将矩阵 A 进行左右翻转再赋给 B ，亦即 $b_{ij} = a_{i,n+1-j}$，而 C=flipud(A) 命令将 A 矩阵进行上下翻转并将结果赋给 C, 亦即 $c_{ij} = a_{m+1-i,j}$ 。D=rot90(A) 将 A 矩阵旋转 90° 后赋给 D ，亦即 $d_{ij} = a_{j,n+1-i}$ ，下面将通过例子来演示各个函数的翻转效果。

例 2.1 假设 A 矩阵为 A=[1,2,3; 4,5,6; 7,8,0]，则调用这三个函数分别将得出如下的结果

```
>>  A=[1,2,3; 4,5,6; 7,8,0]
A = 1     2     3
    4     5     6
    7     8     0
>>  B=fliplr(A)          >>  C=flipud(A)          >>  D=rot90(A)
B = 3     2     1        C = 7     8     0        D = 3     6     0
    6     5     4            4     5     6            2     5     8
    0     8     7            1     2     3            1     4     7
```

- **矩阵乘法**: 假设有两个矩阵 A 和 B ，其中 A 的列数与 B 矩阵的行数相等，则称 A, B 矩阵是可乘的，或称 A 和 B 矩阵维数相容。假设 A 为 $n \times m$ 矩阵，而 B 为 $m \times r$ 矩阵，若 $C = AB$ ，则 C 为 $n \times r$ 矩阵，其各个元素为

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}, \ \text{其中}\ i = 1,2,\cdots,n,\ j = 1,2,\cdots,r \tag{2.4.1}$$

例如下面给出的 A 和 B 矩阵及其乘积矩阵 C 为

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 5 \\ 7 & 8 \end{bmatrix}, \quad C = \begin{bmatrix} 19 & 21 \\ 43 & 47 \end{bmatrix}$$

在 MATLAB 下，矩阵 A 和 B 的乘积可以简单地由运算 C=A*B 求出

```
>>  A=[1, 2; 3, 4]; B=[5, 5; 7, 8]; C=A*B
C =  19     21
     43     47
```

在这里并不需要指定 A 和 B 矩阵的维数。如果 A 和 B 矩阵的维数相容，则可以准确无误地获得乘积矩阵 C，如果二者的维数不相容，则将给出错误信息，通知用户两个矩阵是不可乘的。

- **Kronecker乘法运算**: 若存在两个矩阵 A 和 B, 其中 A 为 $n\times m$ 阶矩阵，B 为 $p\times q$ 阶矩阵，则 A 与 B 矩阵的 Kronecker 乘法运算可以定义为

$$C = A\otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1m}B \\ a_{21}B & a_{22}B & \cdots & a_{2m}B \\ \vdots & \cdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \cdots & a_{nm}B \end{bmatrix} \tag{2.4.2}$$

由上面式子可以看出，Kronecker 乘积 $A\otimes B$ 与 $B\otimes A$ 均为 $np\times mq$ 阶矩阵，但一般情况下 $A\otimes B\neq B\otimes A$。和普通矩阵乘积不同，Kronecker 乘积并不要求两个被乘的矩阵满足任何意义下的维数匹配。Kronecker 积的 MATLAB 命令为 `C=kron(A,B)`，例如给定两个矩阵 A 和 B

$$A=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},\quad B=\begin{bmatrix} 1 & 3 & 2 \\ 2 & 4 & 6 \end{bmatrix}$$

则 A 与 B 的 Kronecker 积可以由下面的 MATLAB 命令求出来

```
>>  A = [1, 2;  3, 4];  B = [1, 3, 2; 2, 4, 6];
>>  C = kron(A, B)
C =  1      3      2      2      6      4
     2      4      6      4      8     12
     3      9      6      4     12      8
     6     12     18      8     16     24
>>  D=kron(B,A)
D =  1      2      3      6      2      4
     3      4      9     12      6      8
     2      4      4      8      6     12
     6      8     12     16     18     24
```

- **矩阵乘方运算**: 一个矩阵的乘方运算可以在数学上表述成 A^x, 而其前提条件要求 A 矩阵为方阵。如果 x 为正整数，则乘方式子的结果可以将 A 矩阵自乘 x 次而得出。如果 x 为负整数，则可以将 A 矩阵自乘 $-x$ 次，然后对结果进行求逆运算就可以得出该乘方结果。如果 x 是一个分数，例如 $x=n/m$, 其中 n 和 m 均为整数，则首先应该将 A 矩阵自乘 n 次，然后对结果再开 m 次方。

矩阵的开方运算是相当困难的，但有了数字计算机，这种运算就不再显得那么麻烦了，用户可以利用计算机方便地求出一个矩阵的方根。

在 MATLAB 环境下，如果给定了一个 A 矩阵，则其乘方矩阵和开方矩阵可以容易地由 `A^x` 求出，其中 `x` 为一个常数。仍考虑下面给出的 A 矩阵

$$A=\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$



该矩阵的平方和 0.1 次幂可以由 A^2 和 A^0.1 两条命令容易地求得

```
>>  A^2                           >>  A^0.1
ans =                             ans =
        30    36    15              0.9750+0.2452i  0.1254-0.0493i  0.0059-0.0604i
        66    81    42              0.2227-0.0965i  1.1276+0.1539i  0.0678-0.1249i
        39    54    69              0.0324-0.1423i  0.0811-0.1659i  1.1786+0.2500i
```

- **MATLAB定义的点运算**: MATLAB 中定义了一种特殊的运算，即所谓的点运算。两个矩阵之间的点运算是该矩阵对应元素的直接运算，例如 C=A.*B 表示 A 和 B 矩阵的相应元素之间直接进行乘法运算，然后将结果赋给 C 矩阵。注意，点乘积运算要求 A 和 B 矩阵的维数相同。这种点乘积运算又称为 Hadamard 乘积。可以看出，这种运算和普通乘法运算是不同的，例如对两个简单矩阵 A 和 B

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix},\quad B=\begin{bmatrix}2&3&4\\5&6&7\\8&9&0\end{bmatrix}$$

来说，普通乘法运算与点乘运算的结果分别为

$$AB=\begin{bmatrix}36&42&18\\81&96&51\\54&69&84\end{bmatrix},\quad A.*B=\begin{bmatrix}2&6&12\\20&30&42\\56&72&0\end{bmatrix}$$

可以看出，这两种乘积结果是不同的，前者是普通矩阵乘积，而后者是两个矩阵对应元素之间的乘积，所以采用点运算时要注意其含义。点运算在 MATLAB 中起着很重要的作用，例如如果 x 是一个向量，则求取函数 x^2 时不能直接写成 x*x，而必须写成 x.*x 。在进行矩阵的点运算时，同样要求运算的两个矩阵的维数一致。其实一些特殊的函数，如 sin() 也是由点运算的形式来进行的，因为它要对矩阵的每个元素求取正弦值。

矩阵点运算不光可以用于点乘积运算，还可以用于其它运算的场合。比如对前面给出的 A 矩阵作 A.^A 运算，则将得出下面的结果

```
>>  A.^A
ans =          1           4          27
             256        3125       46656
          823543    16777216           1
```

在这一例子中，每个矩阵元素都是原来元素的相应乘方，例如其中的数据 $823543=7^7$ 。

- **MATLAB的除法运算**： MATLAB 定义了除法运算，其意义相当于矩阵的求逆运算。更一般地， MATLAB 还定义了矩阵的左除及右除，在这里只给出一般的等效关系，有关矩阵求逆的方法在下章中给出。



* 矩阵的左除： MATLAB 中用 \ 运算符号表示两个矩阵的左除， A\B 即由 Gauss 消去法来获得线性方程 $AX = B$ 的解 X ，亦即 $X = A^{-1}B$ 。如果 A 矩阵不是方阵，也可以求出 A\B ，这时将使用最小二乘解法来求取 $AX = B$ 中的 X 矩阵，用这种方法得出的解可能与使用 pinv() 函数得出伪逆后再求出的结果不一致，所以使用时应该注意。
* 矩阵的右除： MATLAB 中定义了 / 符号，用于表示两个矩阵的右除。 B/A 为 BA^{-1}，但在计算方法上存在差异，更精确地，有 (A'\B')' 。

例 2.2　例如 A 和 B 矩阵由下式给出

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 0\\ 1 & 3 & 5\end{bmatrix},\quad B=\begin{bmatrix}1 & 3 & 5\\ 2 & 4 & 6\end{bmatrix}$$

则由 MATLAB 可以容易地得出 A/B 和 pinv(A')*B' 为

```
>>  A/B
ans =   0.0000    0.5000
       -3.0000    3.5000
      -12.0000   10.2500
        1.0000    0.0000
```

```
>>  pinv(A')*B'
ans =     0.3559    0.3390
         -0.0508    0.2373
          0.0000    0.0000
          0.8475    0.7119
```

可见二者是不同的，所以在使用伪逆时应该注意。由于维数关系，前面给出的矩阵是不能求取 A\B 的。如果使用 MATLAB ，可以容易地求出 A'\B' 如下

```
>>  A'\B'
ans =         0         0
              0    0.2857
              0    0.0000
         1.0000    0.8571
```

* **单个矩阵元素的赋值与运算**: MATLAB 允许用户对一个矩阵的单个元素进行赋值和操作，例如如果想将前面矩阵 A 的第 2 行第 3 列的元素赋为 100, 则可以通过下面语句来完成：

```
>>  A(2, 3)=100
A =    1      2      3
       4      5    100
       7      8      0
```

这时将只改变该元素的值，而不影响到其它元素的值。如果给出的行数或列数大于原来矩阵的范围，则 MATLAB 将自动扩展原来的矩阵，并将扩展后未赋值的矩阵元素置为 0 。例如如果想将前面矩阵 A 的第 4 行第 5 列元素的值定义为 8 ，就可以使用 A(4,5)=8 命令，这样可以得出如下的结果



```
>>  A(4,5) = 100
A =    1      2      3      0      0
       4      5    100      0      0
       7      8      0      0      0
       0      0      0      0      8
```

除了对单个矩阵元素进行定义之外，MATLAB 还允许对子矩阵进行定义和处理。例如可以使用冒号操作符来进行这种定义，下面将通过例子来说明冒号操作符的使用方法。

* A(:,j) 表示 A 矩阵的第 j 列全部元素，而 A(i,:) 将表示 A 矩阵第 i 行的全部元素。
* 冒号表达式的一般形式为 s1:s2:s3，其中 s1 为起始值，s2 为步距，s3 为中止值。使用这样的命令就可以产生一个由 s1 开始，以步距 s2 自增，并中止于 s3 的行向量。例如在仿真中经常使用的时间向量可以通过下面的语句来赋值：

  ```
  t=0:  0.1:  10;
  ```

  通过这一语句将产生一个从 0 开始的，以 0.1 为步距自增的，直到 10 为止的行向量，即产生一个元素为

  ```
  t =  0.0000  0.1000  0.2000  0.3000  0.4000 ...
       9.8000  9.9000 10.0000
  ```

  的行向量，这样的时间向量可以直接用于定步长的数字仿真中。

  如果 s2 不给出，则可以认为自增步距为 1，这样也可以建立起来一个行向量，例如 i=1:10 赋值命令将给出一个自然数的行向量。
* A(1:2, 2:4) 表示对 A 矩阵取第一行和第二行内，并在第二列到第四列中的所有元素构成的子矩阵。如果对前面例子中矩阵 A 作这种操作，则将得到如下结果

  ```
  >>  A(1:2, 2:4)
  ans =    2      3      0
           5    100      0
  ```

  如果用户给出了 A(2:3,1:2:5) 命令，则将得出下面的结果

  ```
  >>  A(2:3, 1:2:5)
  ans =    4    100      0
           7      0      0
  ```

  其含义为取原来 A 矩阵的第二行和第三行，且位于 1, 3, 5 列上的所有元素构成的子矩阵。

由于 MATLAB 提供了丰富的冒号运算，所以用户可以从给出的矩阵中容易地获得子矩阵，这样处理的速度比后面将介绍的利用循环语句来赋值的方式快得多，所以在实际编程时应该尽量采用这种赋值方法。

- **逻辑运算**：除了前面定义的各种运算之外，MATLAB 还支持逻辑运算及表达式处理，例如它使用逻辑运算符 &, | 和 ~ 分别表示“与”，“或”和“非”等逻辑运算。

2.5 MATLAB 的控制语句

和其它高级语言一样，MATLAB 也提供了条件转移语句、循环语句等一些常用的控制语句，从而使得 MATLAB 语言的编程显得十分灵活。MATLAB 支持的控制语句和 C 语言中的控制语句格式是很相似的。下面将介绍一些常用的 MATLAB 语言控制语句及其使用方法与实例。

2.5.1 MATLAB 的循环语句结构

MATLAB 中可以使用两种循环语句：for 语句和 while 语句。这两种语句的基本格式和 C 语言中的循环语句是很相似的，例如 for 语句的基本格式为

```
for  循环变量 = 表达式 1 : 表达式 3 : 表达式 2
     循环语句组
end
```

注意，这里的循环语句是以 end 结尾的，这和 C 语言的结构不完全一致。在 C 语言循环中，循环体的内容是以大括号 {} 括起来的，而在 MATLAB 语言中，循环体的内容是以循环语句和 end 语句括起来的，所以在使用 MATLAB 时应注意这一点。

for 循环体结构的程序框图表示在图 2-6 (a) 中给出，其中 for i=s1: s3: s2 语

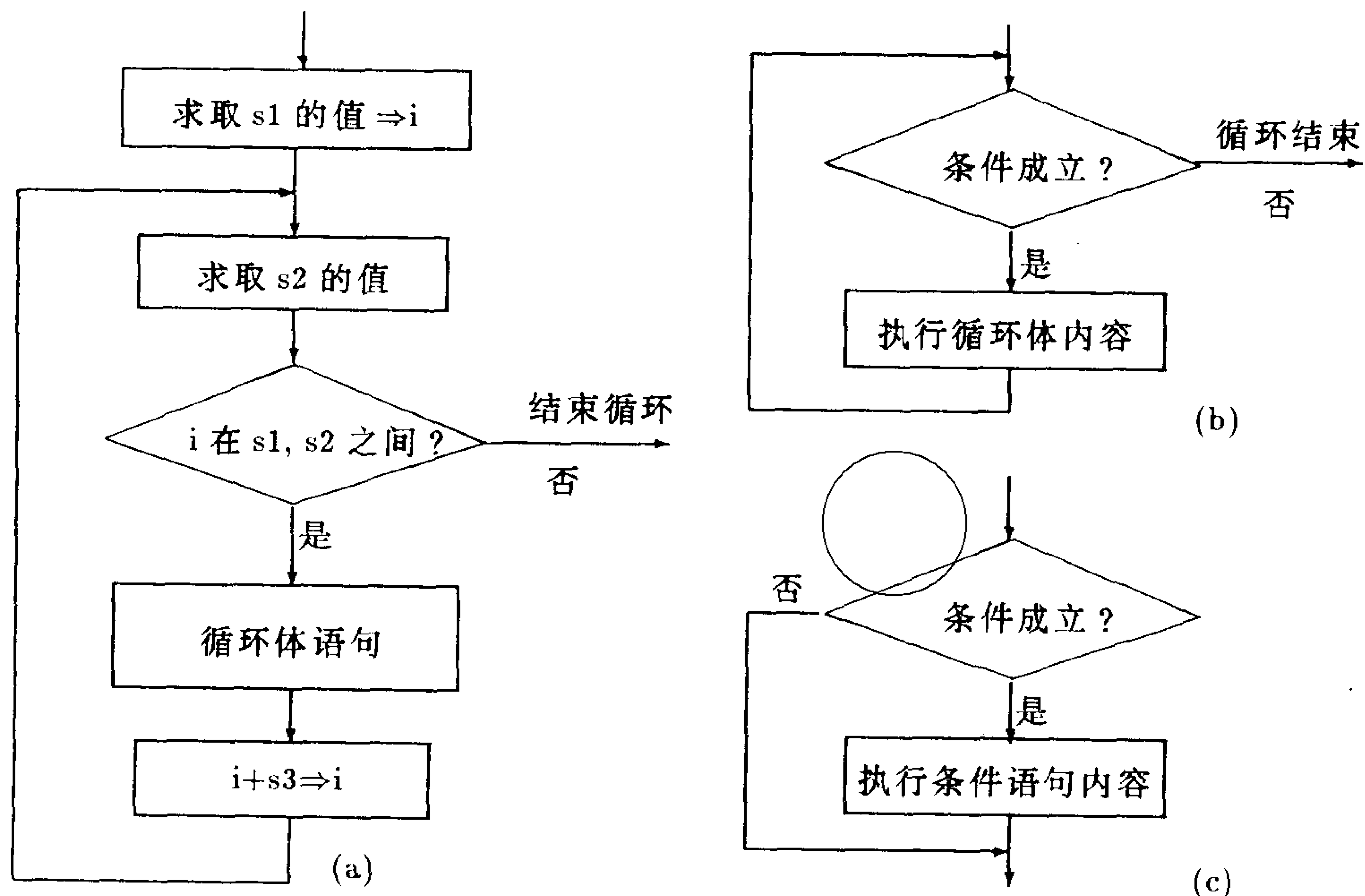


图2-6 MATLAB 下各种控制结果示意图

句的定义在 s3>0 时和 C 语言中的 for (i=s1; i<=s2; i+=s3) 是一致的，而在 s3<0 时它和 for (i=s1; i>=s2; i+=s3) 是一致的。



在 MATLAB 的循环语句基本格式中，循环变量可以取作任何 MATLAB 变量，表达式 1, 2 和 3 的定义和 C 语言相似，即首先将循环变量的初值赋成表达式 1 的值，然后再求取表达式 2 的值，如果此时循环变量的值介于表达式 1 和表达式 2 的值之间，则执行循环体中的语句，否则结束循环语句的执行。执行完一次循环体中的语句之后，则会将循环变量自增一个 s3 的值，然后再判断循环变量是否介于表达式 1 和表达式 2 之间，如果满足仍再执行循环体，直至不满足为止，这时将结束循环语句的执行，而继续执行后面的语句。

例 2.3 如果用户想由 MATLAB 求出 $\sum_{i=1}^{100} i$ 的值，可以作下列的循环：

```
mysum=0;
for i=1:1:100
  mysum=mysum+i;
end
```

在上面的式子中，可以看到 for 循环语句中表达式 3 的值为 1 。在 MATLAB 实际编程中，如果表达式 3 的值为 1 ，则可以在该语句中省略，故该语句可以简化成 for i=1:100 。

在实际编程中，在 MATLAB 下采用循环语句会降低其执行速度，所以前面的程序可以由下面的命令来代替： i=1:100; mysum=sum(i); ，在这一语句中，首先生成了一个向量 i，然后用内部函数 sum() 求出 i 向量的各个元素之和。如果前面的 100 改成 10000 ，再运行这一程序，则可以明显地看出，后一种方法编写的程序比前一种方法快得多。

MATLAB 语言提供了另一种循环语句结构 while ，该循环语句的结构为

```
while  (条件式)
         循环体语句组
end
```

其执行方式为，若条件式中的条件成立，则执行循环体的内容，执行后再判断表达式是否仍然成立，如果表达式不成立则跳出循环，向下继续执行。 while 循环结构的框图表示如图 2-6(b) 所示。

重新考虑例 2.3 ，如果改用 while 循环结构，则可以写出下面的程序

```
sum = 0; i=1;
while (i<=100)
  sum=sum+i;  i=i+1;
end
```

当然， MATLAB 提供的循环结构 for 和 while 是允许多级嵌套的，而且它们之间也允许相互嵌套，这和 C 语言等高级程序设计语言是一致的。

2.5.2 MATLAB 的条件转移语句结构

除了前面介绍的循环语句结构之外， MATLAB 还提供了各种条件转移语句的结构，使得 MATLAB 语言更易于使用。 MATLAB 提供的条件语句最简单的格式是由关键词 if 引导的，其格式为



```
if    (条件式)
      条件块语句组
end
```

其结构的框图如图 2-6 (c) 所示，当给出的条件式成立时，则执行该条件块结构中的语句内容，执行完之后继续向下执行，若条件不成立，则跳出条件块而直接向下执行。

例 2.4　如果将例 2.3 中给出的问题变成求出满足 $\sum_{i=1}^{m} i > 10000$ 的最小 m 。这样就不能直接调用 sum() 函数了。用户可以针对这一问题编写如下的程序段

```
mysum = 0;
for m=1: 1000
   if (mysum>10000), break; end
   mysum = mysum+m;
end
```

注意，这里使用了 break 命令，其作用就是中止上一级的 for 语句循环过程。

while 循环结构在 MATLAB 下也起着相当重要的作用，因为在 MATLAB 下没有提供绝对转移的指令，所以这样的功能就必须通过像 while 那样的循环结构来实现，当然，这样的实现离不开和 if 型条件语句的配合。

例 2.5　如果用户想对一个问题进行回答，而这个问题要求用户键入 y (表示是) 或 n (表示否) 两个字符之一，否则就认为输入的字符有错，程序将要求用户重新输入这样的字符，直至输入 y 或 n 为止，要实现这样的功能，则用户可以由下列的 while 循环来执行

```
ikey = 0;
while (ikey==0)
   s1 = input('Is the answer correct? [y/n]?  ', 's');
   if (s1=='y' | s1=='n') ikey=1; end
end
```

在这一程序段中使用了带有 's' 选项的 input() 函数，其意义是要求用户输入一个字符串。在这一程序段中首先赋给中间变量 ikey 一个初值 0, 然后开始整个循环，在这个循环中用到的条件是 ikey 变量的值为 0，若该条件成立，则继续执行循环体中的各条语句，如果不成立，则跳出循环体而继续向下执行。在这里由于首先假定 ikey=0，所以该条件满足，这样就要求用户输入 s1，然后程序将判断 s1 字符串的值是否为 y 或 n，如果是则将 ikey 的值定义为 1，这样若再向下执行时，则将返回到 while 语句对该条件作判断。显然这时该条件不满足，这将中止此循环体而向下执行，当然这正是我们所期望的。如果输入的 s1 的值不为 y 或 n，则将不对 ikey 的值作任何改动，这样再返回 while 语句去判断条件时可见此条件依然成立，这样将再给出提示并要求输入 s1 字符串，如果用户输入一个新的字符串，则程序将对之重新判断，直至输入了 y 或 n 之后才结束此循环体。这样就能保证输入的字符串为 y 或 n 了，如果不使用附加变量 ikey, 上面的程序段也可以修改成

```
while(1)
   s1 = input('Is the answer correct? [y/n]?  ', 's');
   if (s1=='y' | s1=='n') break; end
end
```

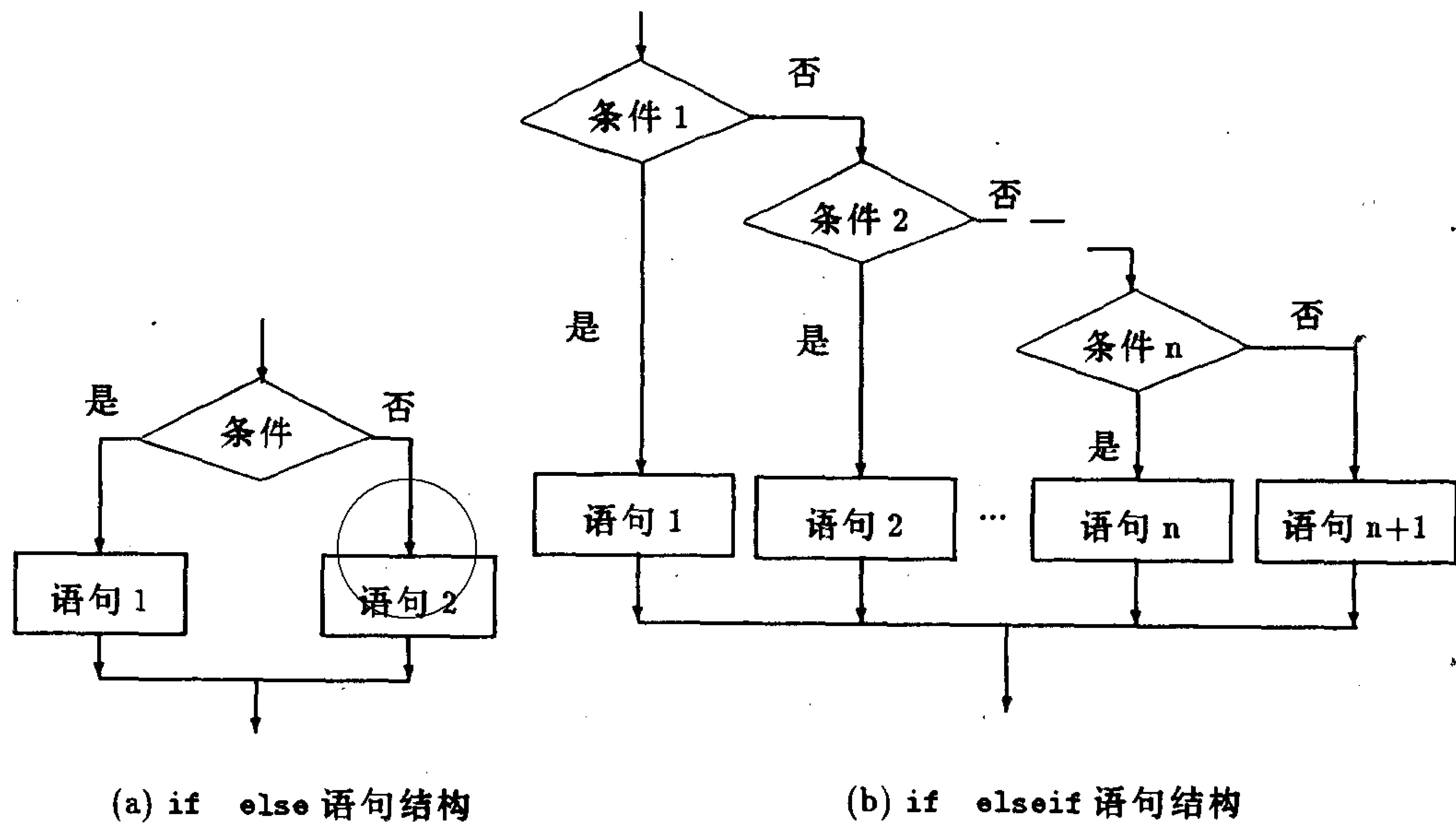



注意在前面的例子中，if 条件式内判断两个表达式是否相等时使用的是 == 符号，而不是 = 号，这和 C 语言的写法是一致的。当然，前面介绍的 if 条件结构只能处理较简单的条件，所以其功能不是很全面。 MATLAB 还提供了其它两种条件结构， if - else 格式和 if - elseif 格式，这两种格式的调用方法分别为

```
if      (条件式)
        条件块语句组 1
else
        条件块语句组 2
end
```

```
if      (条件式 1)
        条件块语句组 1
elseif  (条件式 2)
        条件块语句组 2
...     ...
end
```

其框图表示分别如图 2-7(a) 和 (b) 所示。可见这些语句的结构和功能与其它的程序设计语言 (如 C 和 FORTRAN 是基本一致的) 。下面将通过例子来说明这样条件结构的使用方法。



(a) if else 语句结构　　(b) if elseif 语句结构

图2-7 条件语句结构框图

例 2.6 Newton 迭代法是求解非线性方程的比较常用的方法，如果用户想求出 $f(x)=0$ 的根，且可以求出 $f(x)$ 函数的导数函数 $f'(x)$，这时若用户首先给定初值 x_0，则该方程的根可以由下面的迭代过程求出

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

如果 $f(x)$ 函数为下面的一个多项式

$$f(x)=a_1x^n+a_2x^{n-1}+\cdots+a_nx+a_{n+1}$$

则可以求出该函数的导数函数

$$f'(x)=a_1nx^{n-1}+a_2(n-1)x^{n-2}+\cdots+a_n$$

下面可以用MATLAB的条件语句编写一个程序

```
ikey=0;
n=length(A)-1; B= A(1:n).*[n:-1:1]; i=0;
while (ikey==0)
   xx=x0*ones(1,n+1);
   xv=xx.^[n:-1:0]; xd=xx(1:n).^[(n-1):-1:0];
   C=B*xd';
   if (abs(C)<eps), disp('Deriv is zero'); break;
   else, x1=x0-(A*xv')/C;
      if (norm(x1-x0)<eps), ikey=1;
      else, i=i+1; err= norm(x1-x0);
      disp([i, x1, C, err]); x0=x1;  end
   end
end
disp(['The solution is ', num2str(x1), ', Error=', num2str(err)])
```

其实，MATLAB提供了多项式求值函数polyval()和多项式求导的函数polyder()，它们的调用格式分别为

v=polyval(p, x0) 及 p1=polyder(p)

其中两个函数中输入向量p均为多项式系数降幂排列构成的向量。

在上面的例子中 p=[$a_1,a_2,\cdots,a_n,a_{n+1}$]，polyval()函数中的x0为求值点的x值，该函数将返回 $f(x_0)$ 的值。函数polyder()返回p多项式的导数，亦即p1=[$a_1n,a_2(n-1),\cdots,a_n$]向量。这样前面的Newton法求根函数可以改写成

```
ikey=0;
B=polyder(A);
while (ikey==0)
   C=polyval(B,x0); V=polyval(A,x0);
   if (abs(C)<eps), disp('Deriv is zero'); break;
   else, x1=x0-V/C;
      if (norm(x1-x0)<eps), ikey=1;
      else, err=norm(x1-x0);
      disp([x1, C, err]); x0=x1;  end
   end
end
disp(['The solution is ', num2str(x1), ', Error=', num2str(err)])
```

如果将该段程序存储到polyrt0.m文件中，则执行下面的

```
>>  A=[1, 2, 3, 4, 5, 6]; x0=1;
>>  format long; polyrt0
    0.40000000000000  35.00000000000000   0.60000000000000
   -0.46512062256809  10.28000000000000   0.86512062256809
```




```
-2.08830592947975    2.65509554669349   1.62318530691166
-1.76691305504440   49.77811779384864   0.32139287443535
-1.57071678030152   23.56621889857205   0.19619627474287
-1.49982106430484   14.07123554359790   0.07089571599668
-1.49188785774892   11.55668588650955   0.00793320655592
-1.49179799951876   11.30150883163458   0.00008985823016
-1.49179798813990   11.29864741689345   0.00000001137885
-1.49179798813990   11.29864705458996   0.00000000000000
The solution is -1.492, Error=2.22e-016
```

事实上，MATLAB 提供了一个求解多项式方程的函数 roots()，直接调用该函数可以获得上面例子中的全部根，包括上面迭代得出的实根 -1.4918

```
>> roots(A)
ans = 0.55168546345898 + 1.25334886027721i
      0.55168546345898 - 1.25334886027721i
     -1.49179798813990
     -0.80578646938903 + 1.22290471337441i
     -0.80578646938903 - 1.22290471337441i
```

2.6 MATLAB 的编程基础与技巧

MATLAB 提供了丰富的编程语句结构和实用函数，这往往使得刚刚接触该语言的用户感到无从下手。在本节中将介绍一些常用的编程技巧和方法，以便用户能尽快地启动起来，从而可以更好地使用 MATLAB 语言，使之能发挥其应有的潜能，甚至用户可以在 MATLAB 语言允许的条件下编写出有创造性意义的程序来。

2.6.1 MATLAB 允许的文件类型

由于 MATLAB 本身可以被认为是一种高效的语言，所以用它可以编写出具有特殊目的的程序来，这些程序可以是函数 (又称为 M 函数)，也可以是更广义的文件 (又称为 M 文件)，这些文件都是由纯 ASCII 字符构成的，其后缀名均为 .m 。MATLAB 下的 M 文件可以直接执行，这时用户只需在 MATLAB 提示符 >> 下键入该文件名即可，例如 MATLAB 中给出的演示程序 demo.m 就是这样一个 M 文件。M 函数必须由其它语句来调用，和 M 文件不同，在一般情况下用户不能单独键入其文件名来运行一个 M 函数。MATLAB 下的大多数应用程序都是由 M 函数形式给出的，例如求取矩阵特征多项式的函数 poly() 和求取伴随矩阵的函数 compan() 等。除了 M 函数之外，MATLAB 还提供了大量的底层函数，这些函数称为内部函数，如求取特征值系统的函数 eig()，奇异值分解的函数 svd() 等，这类文件是不可读的。如果没有冲突，在本书中将 M 函数及内部函数均统称为函数。

MATLAB 允许用户调用可执行文件 .exe, 其调用方式是在 MATLAB 提示符 >> 下键入惊叹号！，后面直接跟该可执行文件的文件名或 DOS 命令即可，例如用户可以由



!chkdsk 命令来直接调用 DOS 下的 chkdsk.exe 文件。MATLAB 也允许采用这样的方式来直接使用 DOS 命令，如磁盘复制命令 copy 可以由 !copy 来直接使用，而文件列表命令 dir 可以由 !dir 来调用。事实上为了给用户提供更大的方便，MATLAB 已经把一些常用的 DOS 命令做成了相应的 MATLAB 命令，如在 MATLAB 环境下直接键入 dir 也可以对目录下的相关文件情况进行查看。

MATLAB 还允许用户用 FORTRAN 或 C 语言写出程序，并产生可执行文件，这样也可以用上面的方式直接调用，而 MATLAB 和该程序之间的数据传递是由读写文件的方式来完成的。这种调用格式当然很直观，但其缺点是速度相当慢，此外由于其调用方式的原因，使用起来不是特别规范。

MATLAB 还提供了对 C 或 FORTRAN 语言编写的程序的另一种调用方式，它是通过 MATLAB 提供的 MEX 功能来实现的，在这种方式下，首先要利用 MATLAB 指定的格式来书写 MATLAB 与 FORTRAN 或 C 的接口 (gateway) 程序，然后用它给出的连接程序进行连接，最后获得一个后缀为 MEX 的可执行文件，这种可执行文件的速度较快，因为它和 MATLAB 之间的数据传递是通过指针来完成的，而不涉及到对文件的读写，且其调用格式和 MATLAB 本身的函数调用格式完全一致，此外它还允许在 C 或 FORTRAN 程序下调用 MATLAB 下的任何函数，所以这种方式的应用范围更广。然而使用 C 或 FORTRAN 这样的程序势必会使得用户程序的规模大幅增加，且会使得程序的可读性大大降低，所以不是特别必要，最好不使用 MEX 功能，而应该尽可能地用标准的 MATLAB 语言编写自己的语言程序。

MATLAB 4.0 版要求的对 FORTRAN 和 C 的 MEX 编译程序一般不是很通用的，例如 FORTRAN 语言往往要使用 NDP-FORTRAN 386 (3.0 版)，而 C 语言需要 MetaWare High C (3.0 版) 或 Watcom C/386 (9.0 版)，当然如果想生成特殊的图形界面，还可以采用 Microsoft C (7.0 版) 来生成相应的 DLL 动态连接库程序。

如果在 MATLAB 工作目录下同时有同名的 MEX 文件和 M 函数，则 MATLAB 将优先执行 MEX 文件。MATLAB 的 MEX 程序设计一般涉及较多的内容，在这里就不再赘述了，读者可以参阅文献 [7]。

标准的 MATLAB 数据文件的后缀名为 .mat，这样的文件是一种特殊的二进制编码形式给出的，用户没有必要去读懂这些文件，因为它们可以由 MATLAB 提供的 save 和 load 直接进行处理，后面还将介绍这些语句的调用格式。

在早期的 MATLAB 版本中，还支持图元文件 (后缀名为 .met) 描述的图形，在 MATLAB 4.0 中也保留了该文件类型，但在以后的版本中将逐步取代该文件类型。

2.6.2 MATLAB 工作空间及变量的管理

前面讲过，MATLAB 是以复数矩阵为基本编程单元的一种语言，用户可以直接使用一个矩阵，而不必去显式地指明所用到矩阵的维数。如果用户确实想知道一个矩阵的维数时，则可以使用 size() 函数来测取，该函数的调用格式为

```
[n, m] = size(A)
```




其中 A 为要测试的矩阵名，而返回的两个参数 n 和 m 分别为 A 矩阵的行数和列数。如果等号左边只返回一个参数而不返回两个参数时，则返回的参数是一个 1×2 行向量，其两个元素分别为 A 矩阵的行数和列数。

如果要测试的变量是一个向量而不是矩阵时，当然仍可以由 size() 函数来得出其大小，更简洁地，用户可以使用 length() 函数来求出，该函数的调用格式为

```
n = length(A)
```

其中 A 为要测试的向量名，而返回的 n 为 A 向量的元素个数。如果对一个矩阵作 length() 函数测试，则将返回该矩阵行列的最大值，即该函数等效于 max(size(A))。例如若 A 矩阵是一个 3×4 矩阵，则 length(A) 将返回 4 。

如果用户想查看目前的工作空间中都有哪些变量名，则可以使用 who 命令来完成。例如如果 MATLAB 的工作空间中有 a，b，c，d 四个变量名时，则使用 who 命令将得出如下的结果

```
>>  who
Your variables are:
a          b          c          d
```

如果用户想了解这些变量的具体细节，则可以使用 whos 命令来查看

```
>>  whos
      Name        Size        Elements      Bytes     Density    Complex
         a        1 by 1             1          8        Full        No
         b        3 by 4            12         96        Full        No
         c        1 by 4             4         32        Full        No
         d        4 by 1             4         32        Full        No
Grand total is 21 elements using 168 bytes
```

可见这一命令将列出全部变量的变量名 (Name), 大小 (Size), 元素数 (Element), 字节数 (Bytes), 表现密度 (Density, 有两种方式：完整 Full 与稀疏度), 有无复数 (Complex) 。除了对单个变量给出相应的信息以外，还将给出整个变量空间的占用情况，如这里指出变量空间中总共由 21 个元素单元，占用 168 字节。

用户了解了当前工作空间中的现有变量名之后，则可以调用 clear 命令来删除其中的一些不再使用的变量，这样可以使得整个工作空间更简洁，例如如果用户想删除工作空间中的 a 和 c 变量，则可以使用下面的 MATLAB 命令

```
clear a  c
```

注意，在这一命令下 a 与 c 之间不能加逗号，否则该命令就会被错误地解释成删除 a 变量，然后开始下一个语句 (其内容为 c), 而该语句将被解释成将 c 变量的内容显示出来，这样 c 变量就不再被删除了。如果用户想删除整个工作空间中所有的变量，则可以使用 clear 命令，而在该命令后面不用加任何参数。此外， clear 命令还可以用来删除工作空间中的全局变量、 M 及 MEX 函数的编译结果等。使用 clear 命令时还应该

注意，它们带有的参数必须是在工作空间中现有的矩阵名或函数名，否则将给出错误信息，并中止程序的运行。应该指出的是，MATLAB 下执行一个函数时首先要将其在内存中进行编译，而编译之后如果改动了原来的文件，则在试图执行该文件时仍要执行原来的版本。如果想执行改动后的版本则必须使用 clear 命令来清除原来版本在内存中的编译结果，这时再执行该函数时则会重新在内存中编译。

如果用户想查询在当前的工作空间下是否存在一个变量，则可以调用 MATLAB 提供的 exist() 函数来完成，该函数的调用格式为

```
i=exist(字符串);     例如i=exist('A');
```

返回的值 i 表示 A 存在的形式，例如 i=1 则表示在当前工作空间下存在一个变量名为 A 的矩阵，如果 i=2 则表示在 MATLAB 的工作路径下存在一个名为 A.m 的文件，如果 i=3 则表示在 MATLAB 的路径下存在一个名为 A.mex 的文件，若 i=4 则表示存在一个编译好的名为 A.m 的 SIMULINK 文件，若 i=5 则表示存在一个内部的 MATLAB 函数 A()，除此之外，如果返回的 i 为 0，则表示不存在和 A 有关的变量和文件。

MATLAB 下矩阵默认的显示精度为小数点后面保留 4 位数字，如果数字过大或过小，则将自动地采用科学计数法来表示，这时仍保留小数点后 4 位数字。如果用户想以更高的精度来显示参数，则可以使用 format 命令来控制，该命令的调用格式为

```
format 显示格式控制参数
```

其中显示格式控制参数决定显示的形式，MATLAB 4.0 允许的控制参数如表 2-1 所示。例如如果用户想以高精度来显示结果，则可以使用 format long，若想恢复到原来的默认精度下则键入 format short 即可。当然除了用 format 命令之外，还可以由 MATLAB 命令窗口的 Options | Numeric Format 菜单项来设置，这在前面已经叙述过了。

表2-1　format命令的控制参数表

控制参数	意　　义	控制参数	意　　义
缺省	默认状态，等效于 short	long	15 位有效数字
short	5 位有效数字	short E	5 位有效数字的浮点数
long E	15 位有效数字的浮点数	hex	16 进制格式
+	紧密格式，用 +- 或空格来示意地表示数字	rat	有理格式

2.6.3 MATLAB 的输入与输出语句

MATLAB 的输入与输出函数包括命令窗口的输入与输出及图形界面的输入与输出方法。此外它还允许对文件进行读写。这里将分别介绍各种输入输出方法。

MATLAB 提供了一些输入和输出语句，允许机器和用户之间进行数据交换。如果用户想给计算机输入一个参数，则可以使用 input() 函数来进行，该函数的调用格式为



```
A = input(提示信息, 选项);
```

这里提示信息可以为一个字符串显示，它用来提示用户输入什么样的数据，例如用户想输入 A 矩阵，则可以采用下面的命令来完成

```
A = input('Enter matrix A => ');
```

执行该语句时首先给出 Enter matrix A => 提示，然后等待用户从键盘按 MATLAB 格式输入 A 矩阵。如果在 input() 函数调用时采用了 's' 选项，则允许用户输入一个字符串。

MATLAB 提供的命令窗口输出函数主要有 disp() 函数，其调用格式为

```
disp(A)
```

其中 A 既可以为字符串，也可以为矩阵。如果 A 为一个字符串，其内容为 A='Hello, World'，则其显示的结果为

```
>>  A= 'Hello, World'; disp(A)
Hello, World
```

如果 A 为矩阵，则可以按照下面的形式来显示一个矩阵

```
>>  A=[1, 2, 3; 4, 5, 6; 7, 8, 0]; disp(A)
  1     2     3
  4     5     6
  7     8     0
```

注意，和前面介绍的矩阵显示方式不同，用 disp() 函数显示矩阵时将不显示矩阵的名字，而且其格式更紧密，且不留任何没有意义的空行。

如果用户想将工作空间中的变量保存到文件中，则可以调用 save 命令来完成，该命令的调用格式为

```
save 文件名    变量列表    其它选项
```

注意，和 clear 命令相似，在这一命令中也不能使用逗号，不同的元素之间只能用空格来分隔。例如如果想把工作空间中的 a，b，d 变量存到 mydat.mat 文件中去，则应该给出下面的命令 save mydat a b d，这里将自动地使用文件扩展名 mat 。如果想将整个工作空间中所有的变量全部存入该文件，则应该给出 save mydat 命令。应该指出的是，这样存储的文件是按照二进制的形式进行的，所以得出的文件往往是不可读的，如果用户想按照 ASCII 码的格式来向文件存储数据，则可以在命令后面加一个控制参数 -ascii，该选项将矩阵以单精度的 ASCII 码形式存入文件中去，如果想获得高精度的数据，则可以使用控制参数 -ascii -double 。

MATLAB 提供的 load 命令可以从文件中把矩阵数据调到工作空间中来。该命令的调用格式为

```
load 文件名
```

它的作用是将 save 命令存储起来的数据再从文件中调入工作空间。可见 save 和 load 这一对命令还是相当重要的。

除了对 MATLAB 支持的文件 .mat 提供了标准的打开和存储命令之外，MATLAB 还提供了更低一级的文件打开或处理命令，比如打开文件函数 fopen() 的语句格式为

```
文件句柄=fopen (文件名, 文件类型)
```

其中文件名应该为单引号括起来的字符串，而文件类型可以由一个字符串来描述，其意义和 C 语言的几乎一致，如它可以采用 'r' 来表示一个只读型的文件，而 'a' 表示一个可添加的文件，例如，如果用户想打开一个名为 myfile.xdy 的文件，但不想改变其中的内容，只想从中读出一些数据，则可以把它按一个只读型文件打开，这样就要使用下面的命令

```
myf=fopen('myfile.xdy','r');
```

如果该文件存在，则返回一个句柄 myf，以后就可以对该句柄指向的文件进行直接操作了。例如有了文件句柄之后，就可以调用 fread() 或 fscanf() 等函数从中读取数据，而这些函数首先要用到该文件的句柄，这些函数的使用方法和格式和 C 语言的极其接近。用户还可以调用 fclose(myf) 命令来关闭该文件。如果该文件不存在，则返回的句柄值为 -1，但并不会中断程序的运行。

MATLAB 提供了较实用的字符串处理及转换的函数，例如 int2str() 函数就可以方便地将一个整形数据转换成字符串形式，该函数的调用格式为

```
cStr=int2str(i)
```

其中 i 为一个整数，而该函数将返回一个相关的字符串 cStr。例如在实际计算中如果算出的 Inum 数值为 Inum=15，而在输出中还想给出其它说明性附加信息，则可以给出下面的语句

```
disp(['The value of Inum is ' int2str(Inum) ,'!  OK'])
```

这样该函数将自动地将下面的字符串输出出来

```
The value of Inum is 15!  OK
```

在前面的程序行中，我们在 disp() 函数的变量中构造了一个字符串，该字符串中先将 'The value of Inum is ' 字样原封不动地填写进去，然后将 Inum 的值通过 int2str() 函数转换成相应的字符串，加到前面的字符串后面，最后再将！ OK 字样加到前面的字符串的末尾，这样就形成了一个完整的字符串，最后由 disp() 函数在命令窗口上输出出来。和 int2str() 函数的功能及调用方式相似，MATLAB 还提供了 num2str() 函数，可以将给出的实型数据转换成字符串的表示方式，最终也可以将该字符串输出出来。

MATLAB 还提供了更低一级的函数 sprintf()，其作用是将一些字符和数据按照指定的格式写到一个字符串中，该函数的调用和 C 语言的 sprintf() 的调用几乎一致，这里就不再赘述了，有兴趣的读者可以由 MATLAB 的联机帮助系统获得进一步的信息.



也可以查阅有关 C 语言编程的资料。

除了从命令窗口进行数据交换以外，用户还可以通过图形界面来和计算机进行信息交换，包括提供菜单项、对话框等有关的工具来对一个程序进行干预，有关图形界面的设计与使用的详细情况可以参阅第 8 章。

2.6.4 MATLAB 的联机帮助系统及其应用

由于 MATLAB 提供了大量的函数和命令，如果想记住所有的函数及调用方法一般是不可能的，好在 MATLAB 提供了联机帮助的功能，通过这一功能用户可以容易地获得对想查询的各个函数的有关信息，联机查询可以由 MATLAB 命令 help 来获得，该命令的使用格式为

```
help 命令或函数名
```

它可以直接给出要查询函数的功能和调用方法等信息。例如如果用户想查询有关 lyap() 函数的信息，则可以直接在 MATLAB 提示符 >> 下键入 help lyap 命令，这时将得出下面的帮助信息

```
>>  help lyap
LYAP      Lyapunov equation.
     X = LYAP(A,C) solves the special form of the Lyapunov matrix
     equation:
               A*X + X*A' = -C
     X = LYAP(A,B,C) solves the general form of the Lyapunov matrix
     equation:
               A*X + X*B = -C

     See also DLYAP.
```

从上面的帮助信息就可以看出，它是用来求解 Lyapunov 方程的，它有两种调用方式： X = LYAP(A,B) 和 X = LYAP(A,B,C)，分别对应于两种 Lyapunov 方程。另外，MATLAB 的联机帮助系统还给出了相关的函数，例如这里给出的 See also 条目下指出的参阅 DLYAP 函数，该函数是用来求解离散系统 Lyapunov 方程的。

MATLAB 4.0 允许用户编写自己的帮助信息，这样的帮助信息还可以在 MATLAB 路径下的 Contents.m 文件中描述，用户可以参阅任何一个目录下的该文件。

除了前面所述的命令式查询之外， MATLAB 还提供了 Windows 下的查询方法，这和一般 Windows 程序的联机帮助系统是一致的。例如如果用户选中 MATLAB 界面的 Help | Table of Contents ... 菜单选项，则将出现如图 2-8 所示的帮助界面。可以看出，这里给出的帮助信息就像一本书的目录一样，可以进一步查询的条目都用带下划线的绿色文字给出 (这里由浅色给出)，用户可以用鼠标器点中其中的条目来向下做进一步的查询。除了顺序地查询方式以外，用户还可以按动图 2-8 所示界面中的 Search (查询) 按钮来查询自己所给出的关键词，这时将得出如图 2-9 所示的图形界面，用户可以在最上面

的编辑框中填写出要查询的内容，然后按下 Show Topic (显示) 按钮来找出想了解的问题。这样在没有手册的情况下也可以获得所需的有关信息。

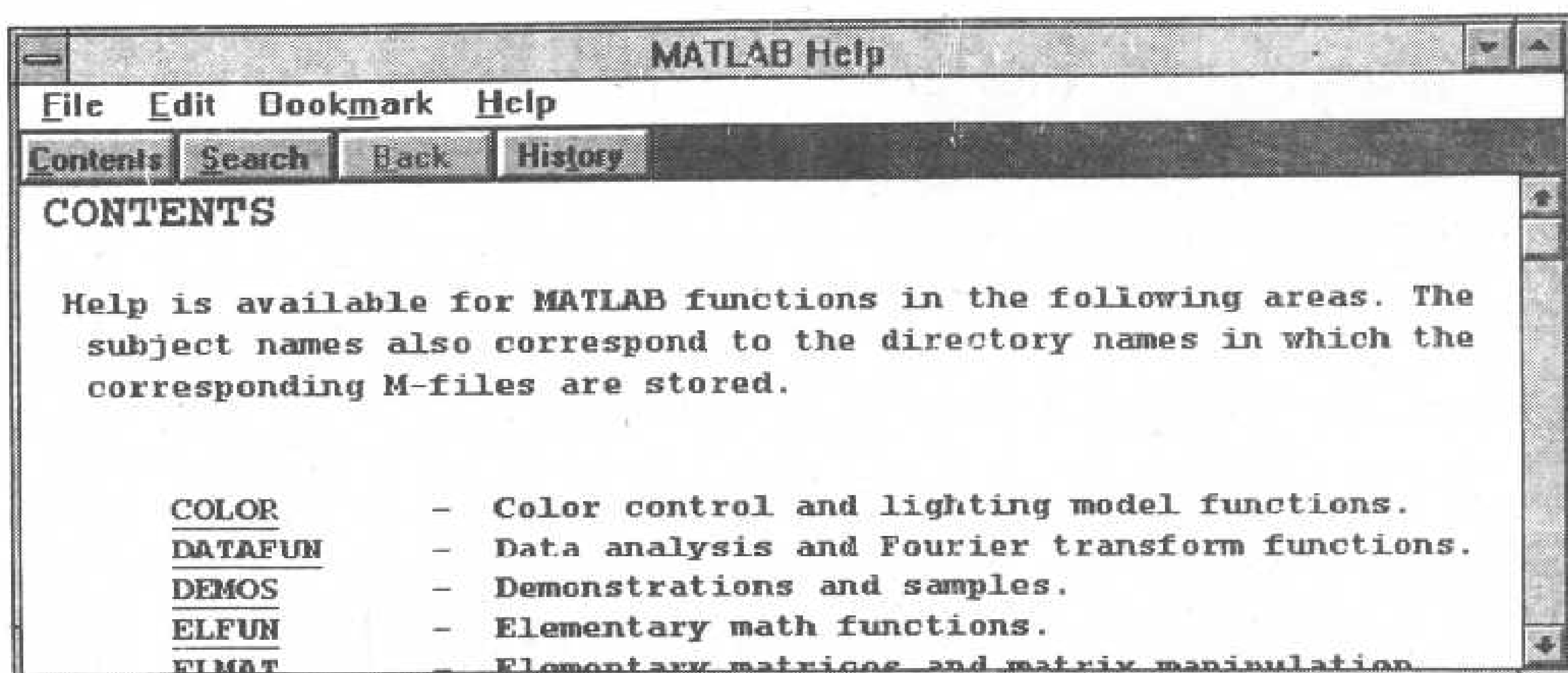


图2-8 MATLAB 函数的联机帮助系统

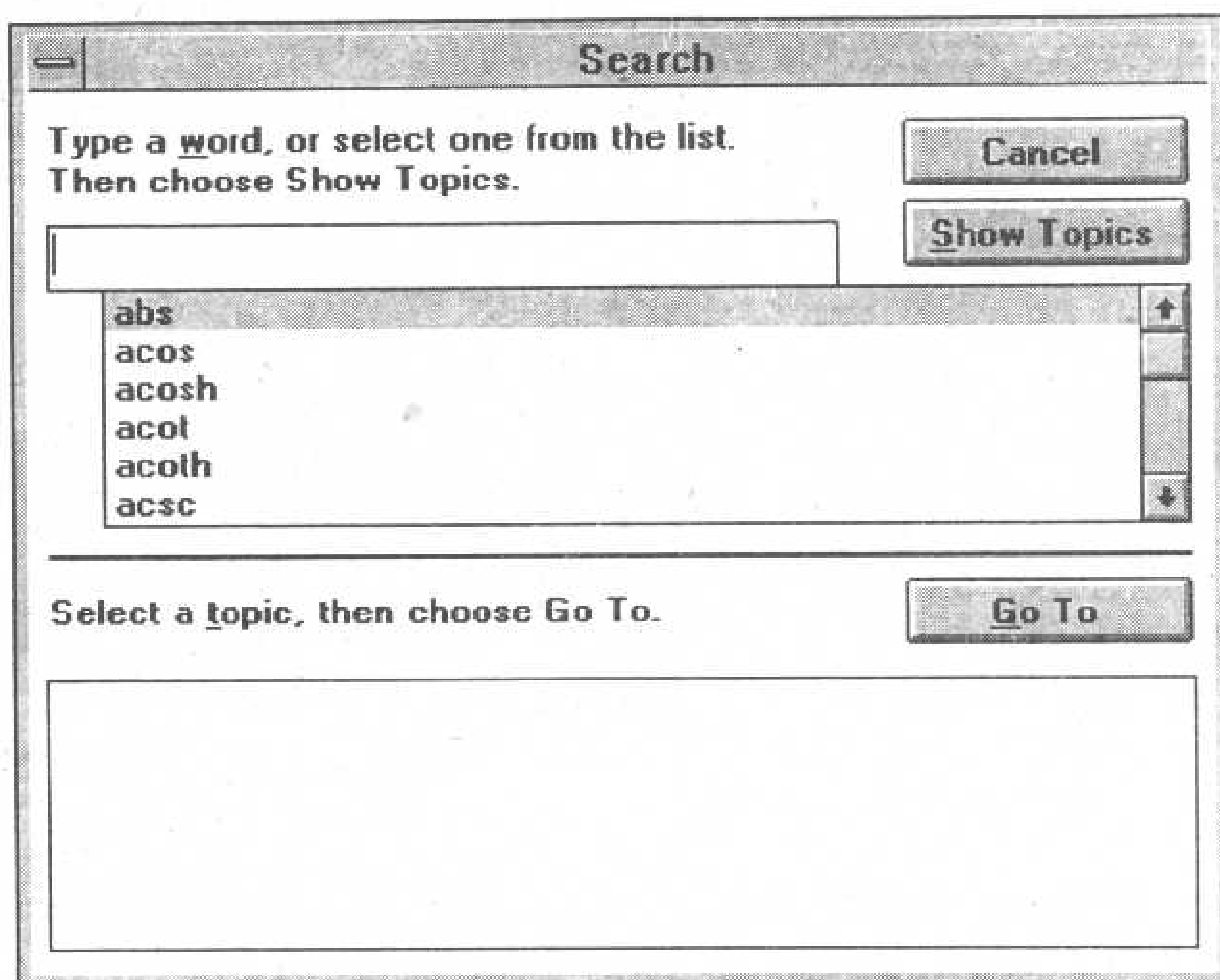


图2-9 MATLAB 联机帮助的查询界面

MATLAB 还提供了关键词查询命令 lookfor，例如若想查出和 decomposition (分解) 有关的 MATLAB 及工具箱函数，则可以给出 lookfor decomposition 命令，并得出如下结果[1]。

```
>> lookfor decomposition
```

[1]注意，由于工具箱安装的多少，在不同的 MATLAB 环境下显示的内容可能有差异。

```
QR        Orthogonal-triangular decomposition.
SCHUR     Schur decomposition.
SVD       Singular value decomposition.
DMPERM    Dulmage-Mendelsohn decomposition of matrix A.
HQR10     Ordered complex Schur decomposition.
SLOWFAST  State-space slow-fast decomposition.
STABPROJ  State-space stable/anti-stable decomposition.
LYAP2     Lyapunov equation solution using eigenvalue decomposition.
SCHORD    Ordered schur decomposition.
```

2.6.5 MATLAB 下 M 文件及 M 函数的编写与调用

MATLAB 下提供了两种文件格式，其中一种是普通的 ASCII 码构成的文件，在这样的文件中只有由 MATLAB 语言所支持的语句，它类似于 DOS 下的批处理文件，这种的文件称作 M 文件，它的执行方式很简单，用户只需在 MATLAB 的提示符 >> 下键入该 M 文件的文件名，这样 MATLAB 就会自动执行该 M 文件中的各条语句。

MATLAB 的另一种，也是最常用的特殊 M 文件称为 MATLAB 函数，这样的函数是由 function 语句引导的，其基本格式如下

```
function 返回变量列表 = 函数名 (输入变量列表)
注释说明语句段
函数体语句
```

这里输入和返回的变量个数分别由 nargin 和 nargout 两个 MATLAB 保留的参数来给出，返回变量如果多于 1 个，则应该用方括号给括起来。输入变量当然应该用逗号来分割。注释语句段的内容如果用户采用 help 命令则可以显示出来，其功能和一般 MATLAB 提供的函数是一致的。

在函数体内使用的除返回和输入变量这些在 function 语句中直接引用的变量以外的所有变量都是局部变量，即在该函数返回之后，这些变量会自动在 MATLAB 的工作空间中清除掉。如果想使得这样的中间变量成为在整个程序中都起作用的变量，则应该把它们设置成全局变量，全局变量是由 MATLAB 提供的 global 命令来设置的。

例 2.7 假设我们想生成一个 Hilbert 矩阵[1)]，该矩阵是一个 $n \times m$ 矩阵，它的第 i 行第 j 列的元素值为 $1/(i+j-1)$。如果想在编写的函数中实现下面几点：

1) 如果只给出一个输入参数，则会自动生成一个方阵，即令 $m=n$。
2) 如果想返回两个参数 A 和 B，则返回的 B 矩阵将为 A 阵的平方，即 $B=A^TA$
3) 在函数中给出合适的帮助信息，包括基本功能、调用方式和参数说明。其实在编写程序时养成一个好的习惯无论对程序设计者本人还是对程序的使用者都是大有裨益的。

下面给出了一个例子，其文件名取作 myhilb.m，并应该放到 MATLAB 的路径下。

```
function [A, B]=myhilb(n, m)
```

[1)] MATLAB 中提供了生成 Hilbert 矩阵的函数 hilb()，这里只是演示函数的编写方法，而在实际使用时还是应该采用 hilb() 函数。

```
% A Demonstrative M-function,  the syntax is
%     [A, B]=myhilb(n, m)
% where
%     n, m are size of the Hilbert matrix, if only one argument
%        given, then a square matrix is generated.
%     A is the Hilbert matrix,
%     B: if two matrices to be returned, B is assigned to A'*A

% Designed by Dr Dingyu XUE, Northeastern University, PRC
%     5 April, 1995, Last modified by DYX at 6 April, 1995
if nargin==1, m=n; end
for i=1: n
   for j=1:m
      A(i,j)=1/(i+j-1);
end, end
if (nargout==2), B=A'*A; end
```

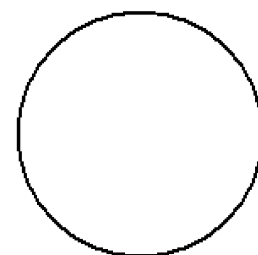
在这段程序中由 % 引导的部分是注释语句，通常用来给出一段说明性的文字来解释程序段落的功能和变量含义等。由前面的第 1) 点要求，首先测试输入的参数个数，如果个数为 1 (即 nargin 的值为 1)，则将矩阵的列数 m 赋成 n 的值，从而产生一个方阵。后面的双重 for 循环语句依据前面给出算法来生成一个 Hilbert 矩阵。最后根据前面的第 2) 点要求判断返回参数的个数，如果个数为 2，则求出 B 矩阵。其实对这个例子来说，即使最后不给出 if 条件语句，返回的结果也是完全一致的。

```
>>  help myhilb
  A Demonstrative M-function,  the syntax is
     [A, B]=myhilb(n, m)
  where
     n, m are size of the Hilbert matrix, if only one argument
        given, then a square matrix is generated.
     A is the Hilbert matrix,
     B: if two matrices to be returned, B is assigned to A'*A
```

注意，这里只显示了程序及调用方法，而没有把该函数中有关作者的信息显示出来。对照前面的函数可以立即发现，因为在作者信息的前面给出了一个空行，所以可以容易地得出结论：如果想使一段信息可以用 help 命令显示出来，则在它前面不应该加空行。

有了函数之后，可以采用下面的各种方法来调用它，并产生出所需的结果

```
>>  [A, B]=myhilb(3,4)
A = 1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
B = 1.3611    0.7500    0.5250    0.4056
    0.7500    0.4236    0.3000    0.2333
    0.5250    0.3000    0.2136    0.1667
    0.4056    0.2333    0.1667    0.1303
>>  [A, B]=myhilb(4)
```





```
A = 1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
B = 1.4236    0.8000    0.5667    0.4413
    0.8000    0.4636    0.3333    0.2619
    0.5667    0.3333    0.2414    0.1905
    0.4413    0.2619    0.1905    0.1507
>>  A = myhilb(4)
A = 1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
```

2.7 MATLAB 的绘图功能

MATLAB 受到控制界广泛接受的另一个重要原因是因为它提供了较方便的绘图功能。在 MATLAB 等软件出现之前，如果想在自己程序中产生一个图形 (即使是二维图形) 是相当困难的，例如如果用户想在自己的 FORTRAN 语言程序中绘制一个图形，首先需要对绘图的数据进行预处理，找出这些数据的最大值和最小值，然后根据它们自动地计算出坐标轴的范围，然后再调用一些绘图命令库函数 (例如著名的 GINO-F) 来把图形在屏幕上显示出来。这样做将耗费程序设计者大量的精力，而且绘制的图形效果往往还取决于设计者的经验，可能不一定令人满意。

此外如果设计者想把这样的程序移植到其它的语言下去实现，例如转移成 C 语言程序，则所有的图形功能部分就必须完全地改写，这可能还要求设计者去调用其它的绘图库函数，这样做对用户来讲可以说是相当苛刻的，也是一个相当沉重的负担。即使不改用其它的高级语言，若想将此程序转移到其它机器上，如从 PC 兼容机转移 Sun SPARC 工作站上，则原来的语句有许多 (尤其是绘图部分) 都需要用户重新进行改写，这样将给使用者带来很多不利的因素。

MATLAB 4.0 允许用户同时打开若干个图形窗口，打开一个图形窗口可以由命令 `figure()` 函数来完成，每产生一个图形窗口，伴随着都有一个图形窗口的句柄。当然用户还可以用 `close()` 函数来关闭该图形窗口。有关句柄和图形窗口特性设置等项内容将在第 8 章中讲述图形界面设计的内容中详细介绍。

2.7.1 MATLAB 下二维图形绘制

MATLAB 等新一代软件和语言使得图形绘制和处理的繁杂工作变得简单得令人难以置信。如果用户将 X 和 Y 轴的两组数据分别在向量 x 和 y 中存储，且它们的长度相同，则可以简单且直观形象地调用 `plot()` 函数，其调用格式为：

```
plot(x, y)
```




这时将在屏幕上绘制出所需要的二维图形。

例 2.8　例如如果用户想绘制出一个周期内的正弦曲线，则首先应该用 t=0:.1:2*pi 命令来产生自变量 t，然后由下面的命令对 t 向量求出正弦向量： y=sin(t); 这样，就可以调用 plot(t,y) 来绘制出所需的正弦曲线，如图 2-10 所示。

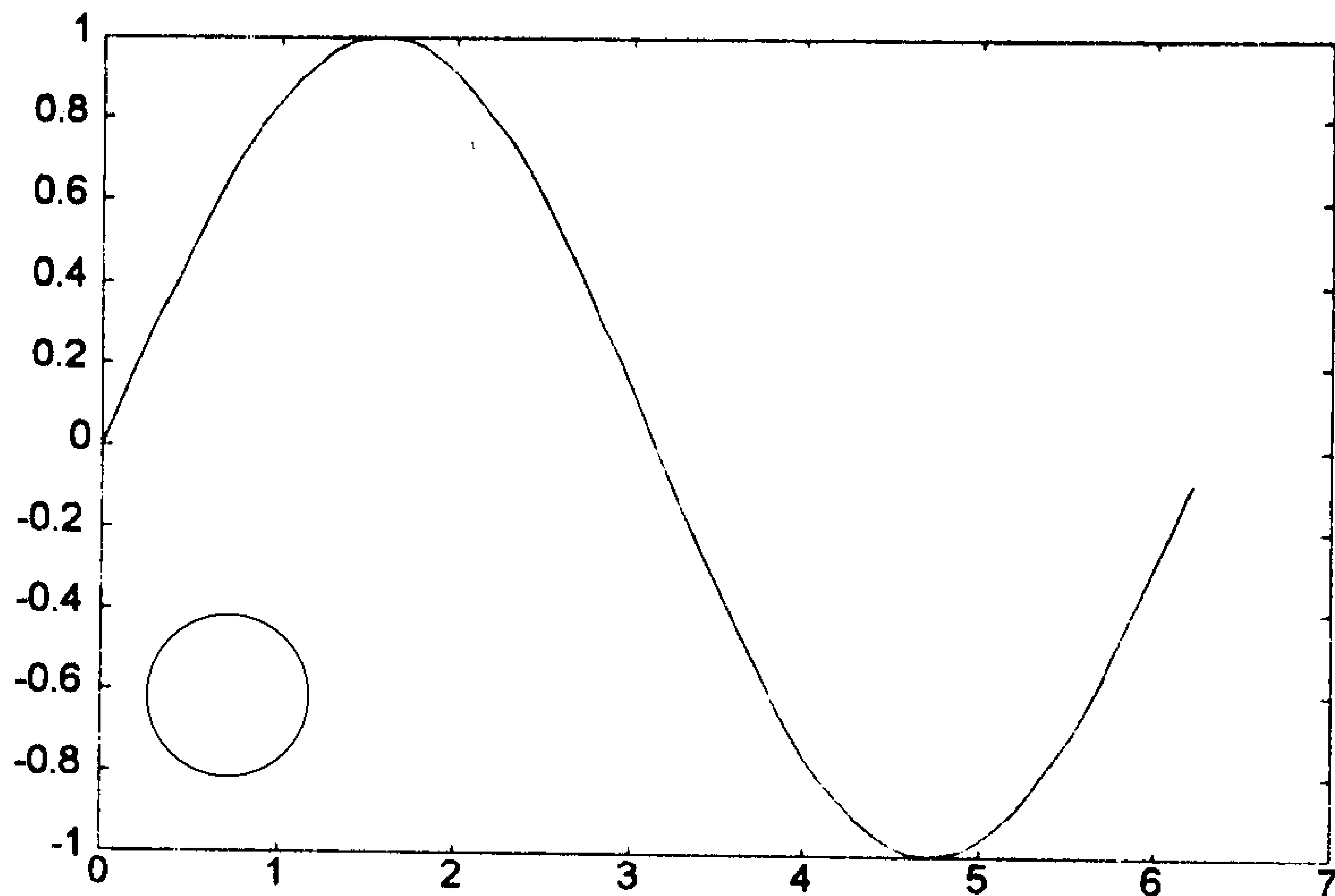


图2-10　MATLAB 绘制的一个周期内的正弦曲线

在 MATLAB 下还允许在一个绘图窗口上同时绘制多条曲线，例如下面的命令

```
>>    t=0:.1:2*pi; y=[sin(t); cos(t)]; plot(t,y)
```

可以产生一组如图 2-11 所示的曲线。这一段语句还是很好理解的， 首先产生一个行向

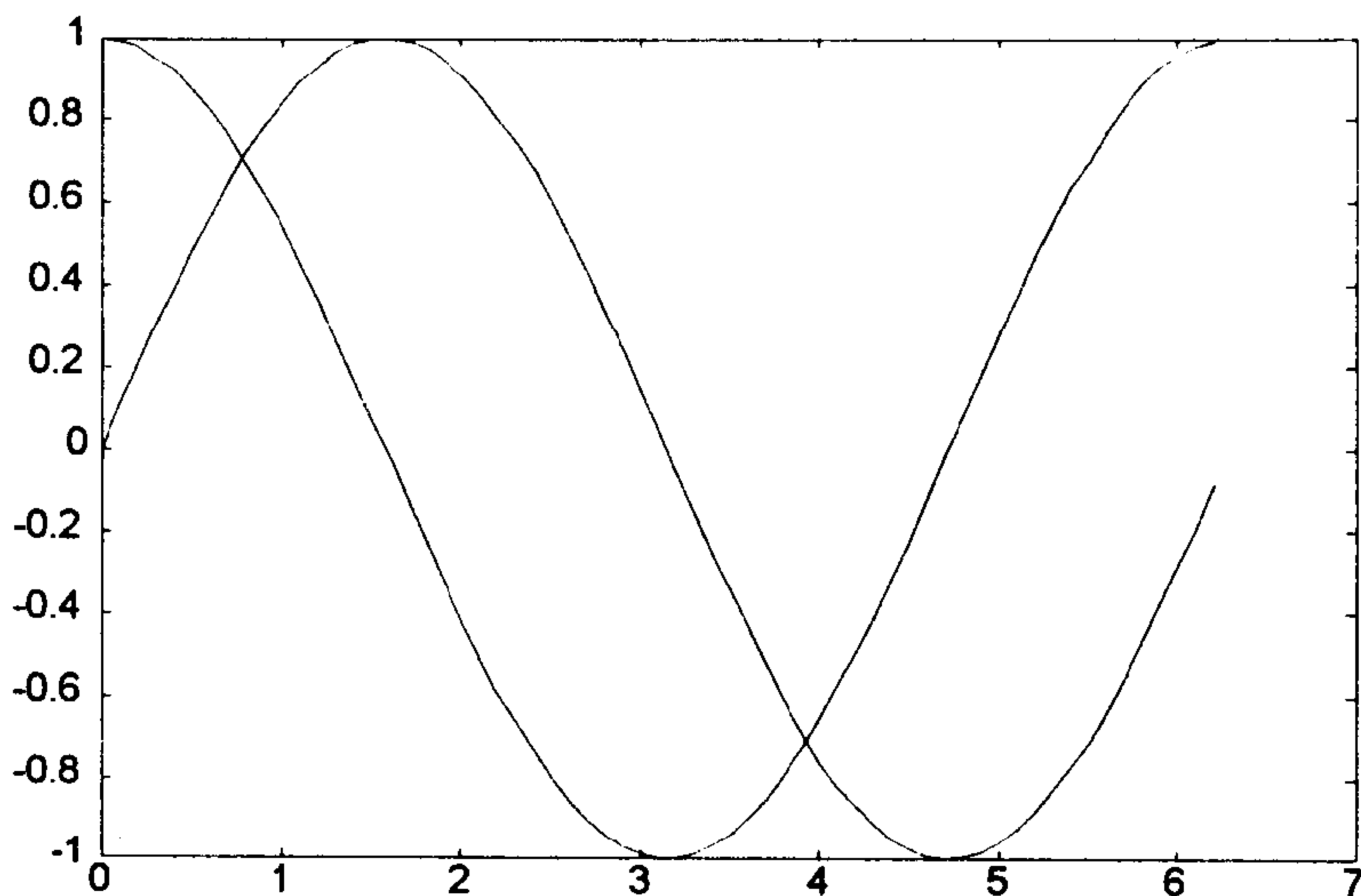


图2-11　MATLAB 绘制的一个周期内的正弦及余弦曲线

量 t，然后分别求取行向量 sin(t) 和 cos(t) 并将它们构成矩阵 y 的两行，最后将两条



曲线在一个坐标系下绘制出来。从图 2-11 可以容易地看出，这两条曲线都是以实线表示的，颜色深浅从图上也基本看不出来。

事实上，在彩色显示器上，MATLAB 会自动地用不同的颜色将图形显示出来，而用单色打印机打印时，也可以用不同的灰度来表示，只是有时区分不是很显然，所以出现难以辨认的情况。同理，利用这样的命令可以在图形窗口上同时绘制出多条曲线，绘制曲线的条数越多，从打印机输出的图上辨认图形的困难也越大。MATLAB 提供了一些绘图的选项，这些选项如表 2-2 所示。

表 2-2 MATLAB 绘图命令的各种选项

选项	意　　义	选项	意　　义
'-'	实线	'--'	虚线
':'	点线	'-.'	点划线
'r'	红色	'g'	绿色
'b'	蓝色	'y'	黄色
'*'	用星号绘制各个数据点	'.'	用点号绘制各个数据点
'o'	用圆圈来绘制各个数据点	'x'	用叉号绘制各个数据点

表 2-2 中给出的各个选项有一些可以连在一起使用，例如选项 '-.g' 表示绘制绿色的点划线。但这样使用时应该注意，有一些选项是不可以连在一起使用的，例如 '-.gx' 将给出 Error in color/linetype argument (颜色 / 线型选项错误) 错误信息，提示用户这里指定的线型是不正确的。这是很好理解的，因为这里用户既要求用点划线来绘制曲线，又指定用叉号来绘制同一曲线，这样从命令本身来分析也是矛盾的，所以将给出错误信息。

带有选项的曲线绘制命令的调用格式为

```
plot(x 轴变量 1, y 轴变量 1, 选项 1, x 轴变量 2, y 轴变量 2, 选项 2, …)
```

在前面的例子中，如果将语句改成

```
>>    plot(t,sin(t),t,cos(t))
```

则将会得出和前面例子相同的曲线。考虑下面的命令

```
>>  t=0:.1:2*pi; y1=sin(t); y2=cos(t); y3=sin(t).*cos(t);
>>  plot(t,y1,'-',t,y2,':',t,y3,'x')
```

分析下面命令，可以看出总共有三条曲线，这三条曲线将分别由实线、点线和叉号表示的曲线，如图 2-12 所示。由于绘制曲线时 x 轴上的点选择得较密集，所以采用叉号选项时每个叉号之间的距离太近，使得它们互相重叠。在一般实际应用中，如果使用叉号来绘制图形时，x 轴上的点之间的距离应该选得比较远。

绘制完曲线后，MATLAB 还允许用户使用它提供的一些特殊绘图函数来进一步修饰画出的图形，例如如果用户在绘制上述的图形后又给出了下面的各条命令

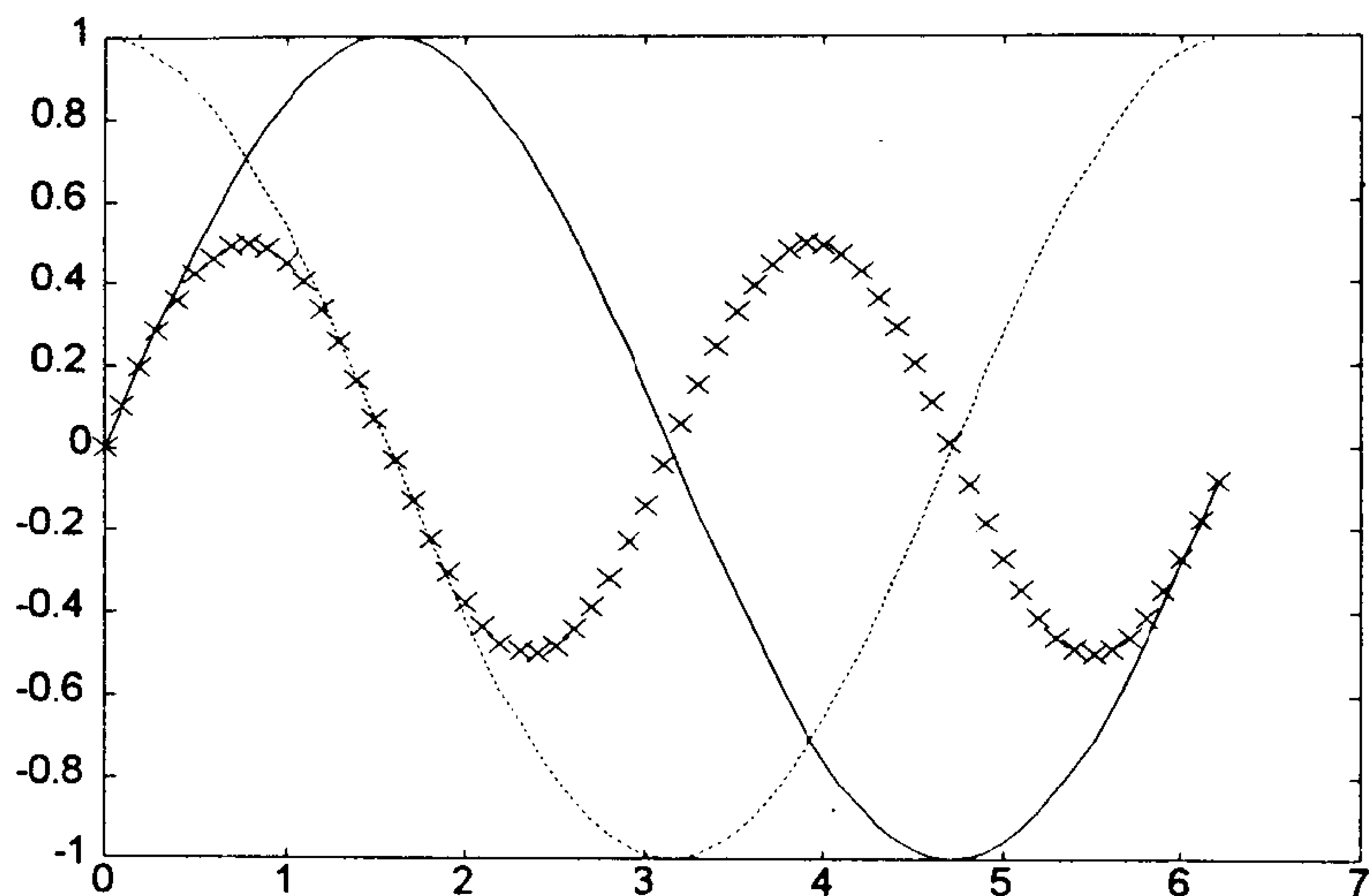


图2-12　以不同线型绘制的三条曲线

```
grid, xlabel('This is my X axis'),
ylabel('My Y axis'), title('My Own Plot')
```

则将给出如图 2-13 所示的图形显示，其中 grid 命令会自动地在各个坐标轴上加上虚线型的网格线，而 xlabel() 和 ylabel() 函数会自动地将其括号中的字符串分别写到图形的 X 轴和 Y 轴上，其中 ylabel() 函数会自动地旋转 90° 来显示。title() 函数则会将其括号中的字符串书写成该图形的标题。

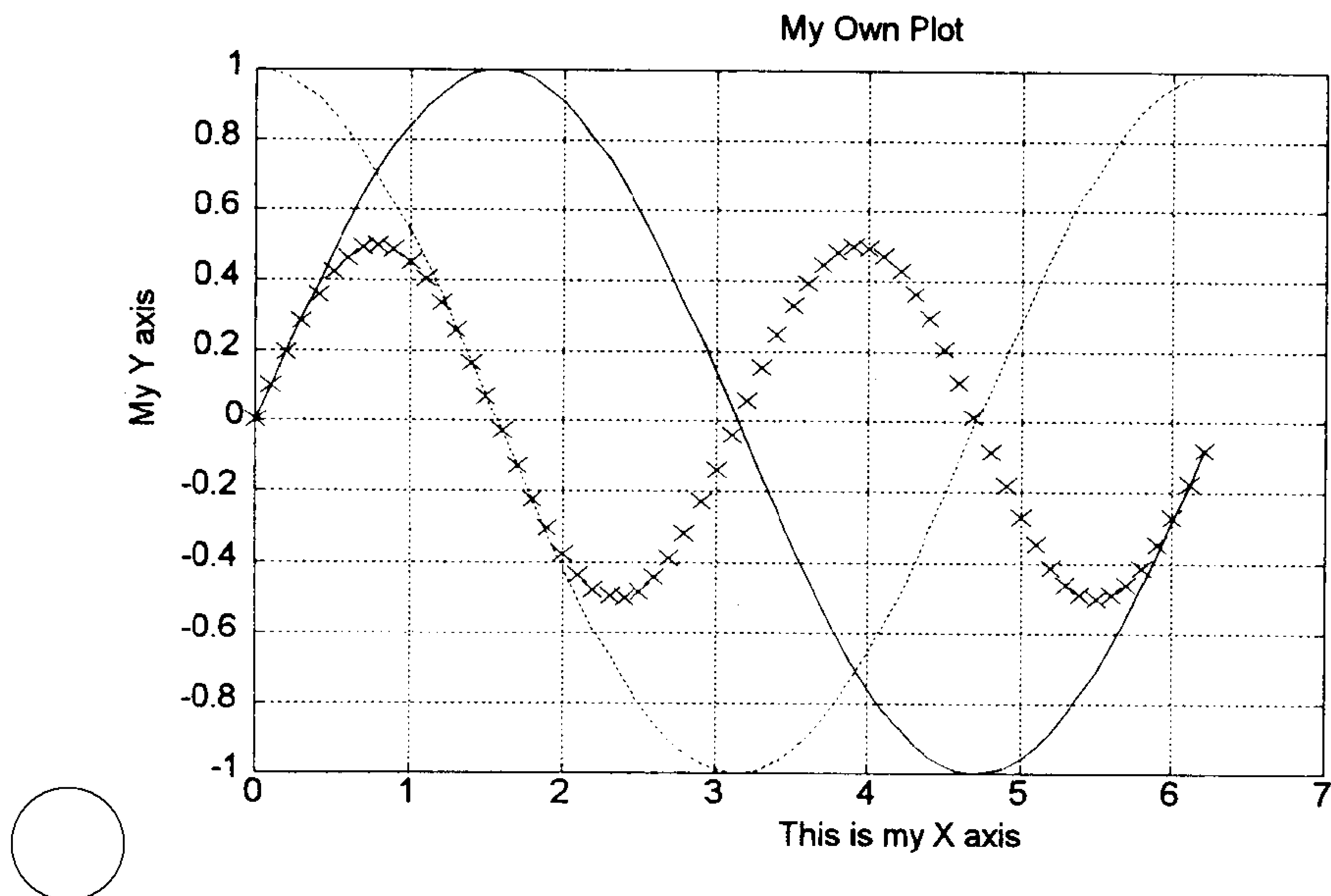


图2-13　MATLAB 图形的进一步修饰

MATLAB 允许用户用鼠标来点选屏幕点，这样的操作是由 ginput() 函数完成的，

该函数的调用格式为

```
[x, y, button] = ginput(n)
```

其中n为选择点的数目，返回的x，y向量分别存储用户点中的n个点的坐标，而button亦为一个n维向量，它的各个分量为鼠标键的标号，如果button(i)=1，则说明第i次按下的是左鼠标键，而该值为2或3则分别对应于中键和右键。这样用户就可以自如地在绘图窗口上用鼠标来选择并操作自己所感兴趣的点了。

除了在标准位置书写标题和轴标志以外，MATLAB还允许在图形窗口的任意位置利用低级命令画直线或书写字符串，这些任务是通过line()和text()函数的调用而完成的，这两个命令的调用格式为

`line(x, y)` 及 `text(x1, y1, chStr,选项)`

其中line()函数在给定的图形窗口上绘制一条由向量x和y定义的折线，而如果采用三参数的调用方式line(x, y, z)，则将在三维坐标系下绘制出一条三维的折线。如果x, y或z的维数不匹配，则将给出错误信息，并中止程序的运行。

例 2.9 如果用户想手动地利用鼠标器绘制出一条折线，则可以配合ginput()函数的调用来完成。例如如果用户想绘制出由10个点构成的折线，则可以由下面的函数完成

```
axis([0, 10, 0, 5]);
[x, y]=ginput(10);
line(x, y)
```

上面的第一个语句用来定义绘图的范围。当然这样绘制图形还有很多显然的毛病，例如在点中屏幕点时没有给出任何标记，用户要求每次点中鼠标左键，则在屏幕上作一个标记。除此之外，用前面的程序时用户也必须点满10个点才能看出效果，如果用户想最多选中10个点，但如果用户按下了鼠标中键或右键则停止选点，而直接绘制出图形，所以对该程序段可以进行下列的改进

```
clg
axis([0, 10, 0, 5])
hold on;  x =[]; y =[];
for i=1: 10
    [x1, y1, button] = ginput(1);
    if (button ~= 1) break; end
    plot(x1, y1, 'o')
    x = [x, x1]; y = [y, y1];
end
line(x, y);  hold off
```

在上面的程序中，为了防止画图时将原来的图形删除，则调用了hold on命令来作一些保护，而图形绘制完成之后，则调用hold off命令来取消这样的保护。在实际使用时，如果用户想将几个plot()命令中建立起来的图形叠印到一起，则可以调用hold命令来进行控制。

MATLAB的text()函数是在指定的点x1，y1处写出一个chStr中给出的字符串，其中x1，y1坐标的单位是由后面的选项决定的，如果后面不给出任何选项，则x1，

y1 坐标的单位和图中的是一致的，如果选项为 'sc'，则 x1，y1 表示规范化的窗口相对坐标，其范围为 0 到 1，亦即该窗口客户区的左下角坐标为 (0,0)，而右上角的坐标为 (1,1)。这样用户就可以在自己选定的位置上写出一个字符串了。

例 2.10 如果用户想在鼠标按下的位置写出一个字符串来给出鼠标点的坐标值，则首先应调用 ginput()，然后在点中的位置输出一个含有该位置信息的字符串，则可以提供下面的程序段来完成

```
clg
axis([0, 10, 0, 5])
hold on;  x =[]; y =[];
for i=1: 10
    [x1, y1, button] = ginput(1);
    if (button ~= 1) break; end
    chStr = ['(' num2str(x1), ',' num2str(y1) ')'];
    text(x1, y1, chStr);
 end
```

如果用户将上述的程序段键入，则就会通过实际的运行来体会该程序段中每个命令的意义了。

2.7.2 MATLAB 下特殊坐标图形

除了前面介绍的二维图形之外，MATLAB 还允许绘制极坐标曲线、对数坐标曲线和直方图等，而这些曲线在一些特定的场合都是很重要的。下面将分别介绍极坐标曲线、直方图和对数 (或半对数) 坐标曲线的绘制函数。

在控制系统的分析中，如果要求绘制出一个系统的 Nichols 曲线，则需要采用极坐标的方式来绘制图形。MATLAB 中提供了一个极坐标曲线绘制的函数 polar()，该函数的调用格式为

```
polar(theta, rho, 选项)
```

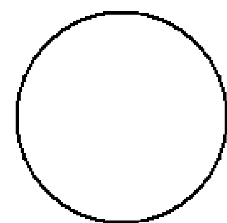
其中 theta 和 rho 分别为角度向量和幅值向量，MATLAB 要求 theta 和 rho 的长度相同，而选项的内容和 plot() 函数的基本一致。

例 2.11 考虑下面的极坐标模型

$$\rho = \cos\left(\frac{5\theta}{4}\right) + \frac{1}{3}$$

如果选定 θ 的范围为 $[0, 8\pi]$，则可以键入下面的命令

```
>>  theta = 0: 0.1: 8*pi;
>>  polar(theta, cos(5*theta/4)+1/3)
```



上面的命令将绘制出如图 2-14 所示的极坐标曲线。

MATLAB 下还给出了直方图的绘制函数 bar()，该函数还可以用来将原来的二维数据变换成直方图所用的数据，以便可以使用 plot() 函数直接绘制出来。bar() 函数的调用格式为



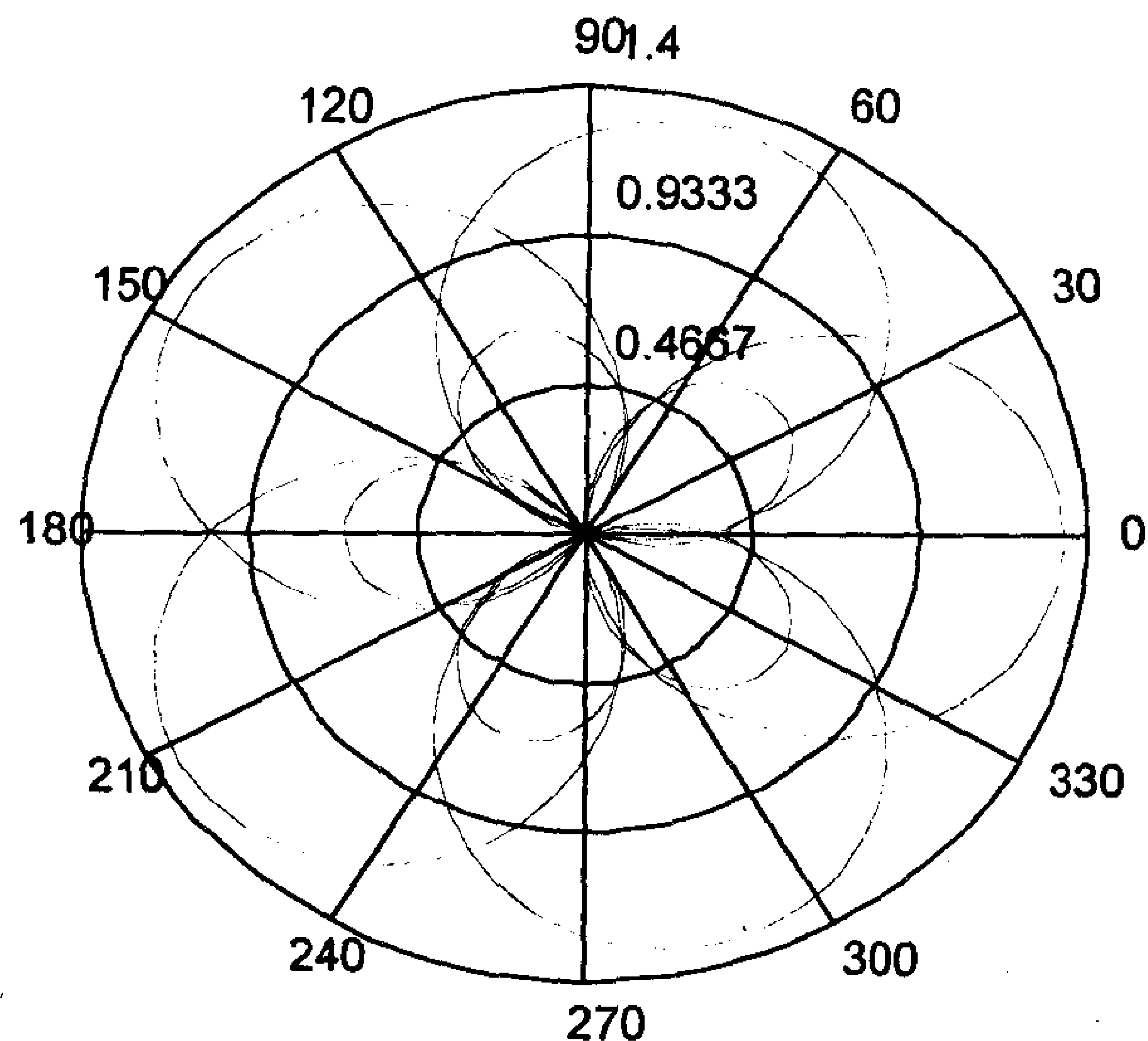


图2-14 极坐标图形的绘制

bar(x, y, 选项); 或 [xx, yy] = bar(x, y);

其中前一种调用方法将直接由给定的 (x,y) 向量绘制出直方图，其使用的方式与 plot() 函数是类似的，也可以使用一些选项来控制图形的线型与颜色，后一种方式下把给出的普通图形数据矩阵对 (x, y) 转换成直方图所需要的 (xx, yy) ，但在这种调用方式下并不输出图形。这样变换而得出的 (xx, yy) 向量可以直接由 plot(xx, yy) 函数来绘制，这时它的作用和第一种调用方式一致。

例 2.12 将一个周期内的正弦值在两种步长下可以用 bar() 函数绘制出来，其程序如下

```
t1 = 0: 0.2: 2*pi;  y1 = sin(t1);
t2 = 0: 0.5: 2*pi;  y2 = sin(t2);
bar(t1, y1); axis([0, 2*pi, -1, 1]);
hold on
[t3, y3] = bar(t2, y2);
plot(t3, y3); hold off
```

可以看出，对不同的采样步长 0.2 和 0.5 ，可以求出两个序列 y1 和 y2, 从而可以绘制出两条直方图曲线。注意，这两条直方图是分别用上面介绍的两种方法绘制的，绘制出来的曲线由图 2-15 所示。

MATLAB 还允许在一个图形窗口上绘制多个图形，例如它可以将窗口分割为上下两个部分，然后在该窗口上分别绘制出系统的幅频曲线和相频曲线的 Bode 图，对于多变量系统来说，还可以将一个窗口分割成 $m \times m$ 个部分，这样在一个图形窗口下就可以绘制出多变量系统的逆 Nyquist 曲线。分割图形窗口的工作是由 MATLAB 提供的 subplot() 函数来设置的，该函数的调用格式为

subplot(n, m, k)

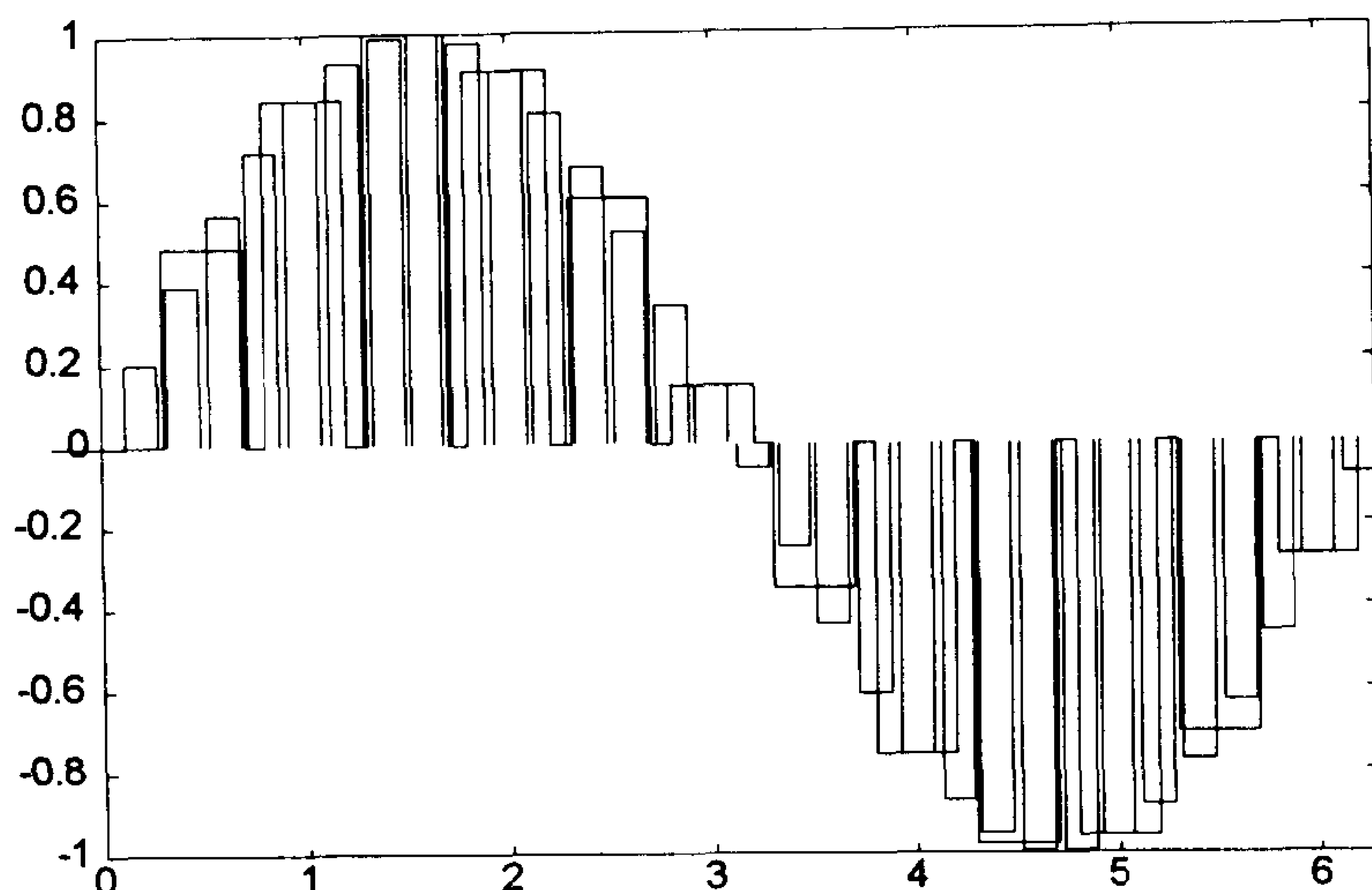


图2-15 直方图的绘制

其中 n，m 分别表示将这个图形窗口分割行列数，而 k 表示要画图部分的代号，例如如果想将窗口分割成 4×3 个部分，则右下角的一块的代号为 12，这时如果 subplot(4,3,6) 则表示想要在该块上绘制图形。MATLAB 最多允许 9×9 的分割。在 MATLAB 下允许每个绘图部分以不同的坐标系单独绘制图形。

在实际应用中，由于数据分布的原因经常需要用对数坐标来表示，例如在控制系统分析中经常需要绘制 Bode 图等，则需要用对数或半对数坐标来完成，MATLAB 中提供了一些对数和半对数坐标曲线绘制的函数，这些函数的名称及调用格式分别为

```
semilogx(x, y, 选项), semilogy(x, y, 选项), loglog(x, y, 选项)
```

其中选项参数的定义与 plot() 函数的完全一致，所不同的是坐标轴的选取。从各个函数的名称上很显然 semilogx() 只对横坐标进行对数变换，而纵坐标仍保持线性坐标，而 semilogy() 只对纵坐标进行对数变换，而横坐标仍保持线性坐标，loglog() 则分别对横纵坐标都进行对数变换，最终得出全对数坐标的曲线来。

例 2.13 在这里给出图形窗口的分割方法，并介绍几种图形的绘制方法，基于这一目的编写了下面的一小段程序

```
theta = 0: 0.1 : 6*pi;  r = cos(theta/3)+1/9;
subplot(2,2,1), polar(theta, r);
subplot(2,2,2); plot(theta, r);
subplot(2,2,3); semilogx(theta, r); grid
subplot(2,2,4); semilogy(theta, r), grid
```

在后两个曲线上，还对坐标加了网格。这样得出的图形如图 2-16 所示。

和线性坐标向量的选取不同，在 MATLAB 下还给出了一个实用的函数 logspace() 按对数等间距地分布来产生一个向量，该函数的调用格式为

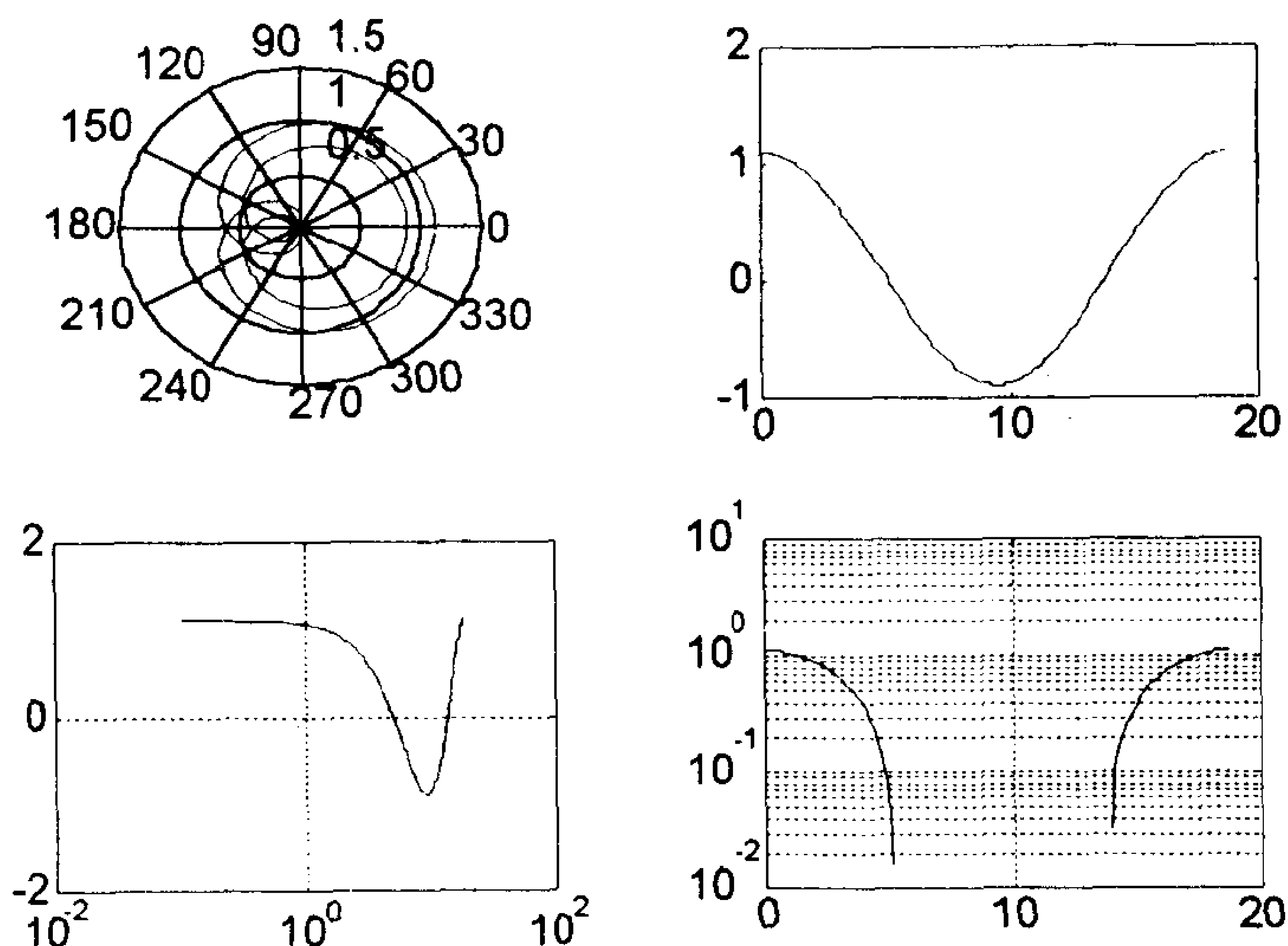


图2-16　绘图窗口的分割与不同图形绘制

```
x=logspace(x1, x2, n)
```

其中 x1 和 x2 分别表示向量的起点和终点，而 n 表示需要产生向量点的个数，这一参数一般也可以不给出，这时将采用其默认值 n=50 。在控制系统频域分析中最好采用这样的方法来构成频率向量。

除了前面介绍的各种曲线类型绘制方法以外，MATLAB 还提供了大量的其它实用图形绘制函数，如 stem(), stairs(), errorbar(), hist(), rose(), compass(), feather(), comet()，这些函数的作用和使用方法很难由几句话讲清楚，但如果用户由一些数据来直接调用这些函数，则它们的作用就很容易显示出来了，用户可以根据自己的实践充分体验这些特殊函数的调用方法。

2.7.3 MATLAB 下图形对象的修改

和其它传统的软件不同，MATLAB 可以自动根据要绘制曲线数据的范围选择合适的坐标系，使得曲线能够尽可能清晰地显示出来，所以一般情况下用户不必担心绘图坐标的选择。但是如果用户觉得自动选择的坐标不合适，则可以用手动的方式来选择新的坐标系，手动变换坐标系的工作可以由 axis() 函数的调用来完成，该函数的调用格式为

axis([$x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\min}$, $z_{\max}$])

从这一命令可以看出，在这里用户可以自由地指定 x, y 轴，甚至 x, y, z 轴的坐标范围。如果用户只给出了四个参数，即 $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$，则将分别按照用户给出 x, y 轴的最小值和最大值选择坐标系，以便绘制出合适的二维曲线。如果除了前面的四个参数之外用户还指定了 $z_{\min}$, $z_{\max}$，则 MATLAB 在绘制三维曲线时会参照用户指定的三个坐标轴的范围来绘制出最终的图形来。

MATLAB 4.0 和其它软件或早期版本的最大区别在于，它在图形绘制时其中每个图形元素 (比如其坐标轴或图形上的曲线、文字等) 都是一个独立的对象，用户可以对其中任何一个图形元素进行单独地修改，而不影响图形的其它部分，具有这样特点的图形称为向量化的绘图方法。这种向量化的绘图要求给每个图形元素分配一个句柄 (handle)，以后再对该图形元素作进一步操作时，则只需对该句柄进行操作即可。例如， plot() 函数将返回一个句柄，这样其调用方式为

```
h=plot(x,y)
```

这样在画出 x, y 曲线的同时，将该曲线的句柄赋给 h，这样以后如果想改变图形的颜色等参数，则需要再调用 set() 函数来改变其属性，其中 set() 函数的调用格式为

```
set(句柄, 属性 1, 属性值 1, 属性 2, 属性值 2,···)
```

其中句柄可以为当前曲线的句柄，如由这里的 plot(x,y) 命令而构成的曲线句柄 h，当然也可以是其它句柄，比如说窗口的句柄或坐标轴的句柄等。

如果想改变 h 句柄所对应曲线的颜色，则可以调用下面的命令

```
set(h,'Color',[1,0,0]);
```

这里对 'Color' 参数进行赋值，将该曲线变成红色 (由 [1,0,0] 决定)，当然还可以对其它参数进行修正，其实，像 line() 和 text() 这样的函数也可以返回句柄。此外用户还可以对图形窗口的底色进行修正，这是通过调用

```
set(gcf, 'Color',[1,1,1])
```

函数来将图形窗口的颜色设置成白色。有关图形窗口进一步设置的详细内容请参见第 8 章。

除了对曲线和窗口的属性进行修改以外， MATLAB 还提供了对坐标轴的修改方法，当前的坐标轴句柄是由 gca 函数来获得的，如果用户想把当前的 X 坐标轴的颜色变为绿色，则可以给出下面的命令

```
set(gca,'XColor',[0 1 0])
```

由上面的命令可以看出，用户可以随意地改变坐标轴的颜色，相应地，用户还可以使用属性 'Ycolor' 和 'ZColor' 来分别地改变 Y 和 Z 坐标轴的颜色。如果用户直接使用 'Color' 属性，将修改由坐标轴框起来的部分的颜色。坐标轴属性的设置命令为

```
set(gca, 属性 1, 属性值 1, 属性 2, 属性值 2, ···)
```

其中一些常用属性简述如下：

- Box **属性**: 决定图形坐标轴是否为方框形式，选项为 'on' (有方框) 或 'off' (无方框)。
- ColorOrder **属性**：设置多条曲线的颜色顺序，默认的矩阵为：

  [1 1 0; 1 0 1; 0 1 1; 1 0 0; 0 1 0; 0 0 1]

  亦即对应的颜色顺序分别为黄、粉、天蓝、红、绿和蓝色，用户可以修改该矩阵的参数来改变曲线颜色的顺序设置或添加或删除其它颜色，例如用户可以采用如下命令



```
set(gca,'ColorOrder',[1 1 0; 0 1 1; 1 0 1])
```

来设置三种曲线的颜色，其顺序为黄、天蓝和粉色。

- **坐标轴方向属性**: Xdir，Ydir，Zdir 分别设定 x, y, z 坐标轴的方向，其选项为 'normal' (正常，默认) 或 'reverse' (反向)，例如若 Zdir 设置成 'reverse' 则表示 z 轴的方向是从上到下的。
- **坐标轴颜色和线型属性**: XColor，YColor，Zcolor 分别设置 x, y, z 坐标轴的颜色，LineWidth 是描述线的宽度的数值，Xgrid，Ygrid 和 ZGrid 分别控制在 x, y, z 坐标轴上是否加网格，其值可以取作 'on' 和 'off'。
- **坐标轴的标尺属性**: Xtick，Ytick 和 Ztick 都是向量，分别表示 x, y, z 坐标轴上的标度的位置，而 XTickLabels，YTickLabels 和 ZTickLabels 是和标度同样大小的向量，分别表示 x, y, z 轴上标度的符号。
- **字体设置属性**： FontAngle 设置字体的角度，其选项为 'normal' (正常，为默认值)、'italic'(斜体) 或 'oblique'(倾斜); FontName 设定字体名称，可以取为 'Helvetica' 和 'Times New Roman' 等；FontSize 设置字号的大小，其单位为 pt (1 英寸 =72.27pt); 其它常用的字体选项为 FontStrikeThrough (选项为 'on' 或 'off'), FontUnderline (选项为 'on' 或 'off'), FontWeight (选项为 'light'，'normal'，'demi' 或 'bold'), 这些选项及意义和 C 语言 Windows 编程中的含义是一致的，在这里就不赘述了。
- **单位制属性** Units: 在坐标轴设定中可能要用到很多尺度的数值，这些数值的单位是由这里的单位制来设定的，可以选择的值为 'inches' (英寸), 'centimeters' (厘米), 'normalized' (归一值), 'points' (点数 pt) 或 'pixels' (像素)。

例 2.14 如果用户给出下面的 MATLAB 命令

```
>> set(gca, 'XColor',[1 0 1],'YColor','g','FontName','Times New Roman',...
      'Ydir','reverse');
```

则会自动地将 x 轴线的颜色设置成粉色，并将 y 轴线的颜色设置为绿色，后面的命令将把坐标系的字体选择为 Times New Roman 字体，而 y 轴的方向被设置成从下到上。可见用户可以通过简单的命令就把坐标轴设置成希望的样子，而同样的设置在 C 语言下还是需要很多条命令来完成的，且没有上面介绍的那样直观，并且它需要用户将改变后的源程序进行重新编译和连接。从这一简单的例子也可以看出 MATLAB 语言的高度集成性与强大的功能。

MATLAB 还允许在一个图形上采取不同的坐标系，这可以由 axes() 函数来完成。由于篇幅所限，无法详细介绍该函数，用户可以自己去查询或键入 set(axes) 来显示出相关的信息。

2.7.4 MATLAB 三维图形绘制

MATLAB 的三维图形绘制也是很有特色的。在早期的版本中，可以由 mesh() 绘制出三维图形，但是该函数有很多弊病，例如，由该命令绘制出的三维图形没有对 x, y, z

坐标的任何标度，所以绘制出来的图形并没有很多实际的用途。在 MATLAB 4.0 版本中对三维图形的绘制做了极大的改进，不仅绘制出来的图形上有三个坐标的标尺，而且还允许由多种方式来绘制三维图形。

首先，和原来的二维图形相对应， MATLAB 提供了 plot3() 函数，它允许用户在一个三维空间内绘制出三维的曲线，该函数的调用格式为

```
plot3(x, y, z, 选项)
```

其中 x，y，z 分别为维数相同的向量，分别存储曲线的三个坐标的值，而这里所使用的选项和 plot() 是一致的，它可以定义曲线的线型、颜色等信息，具体的可以参见表 2-2 。

例 2.15　假设有一个时间向量 t, 对该向量进行下列运算则可以构成三个坐标的值向量

$$x=\sin(t),\quad y=\cos(t),\quad z=t$$

则如果想用绿色的实线绘制此图形，就应该键入下面的程序段

```
t = 0: pi/50: 10*pi;
x = sin(t); y = cos(t); z = t;
plot3(x, y, z, 'g-')
```

由上面程序段绘制出来的三维曲线如图 2-17 所示。

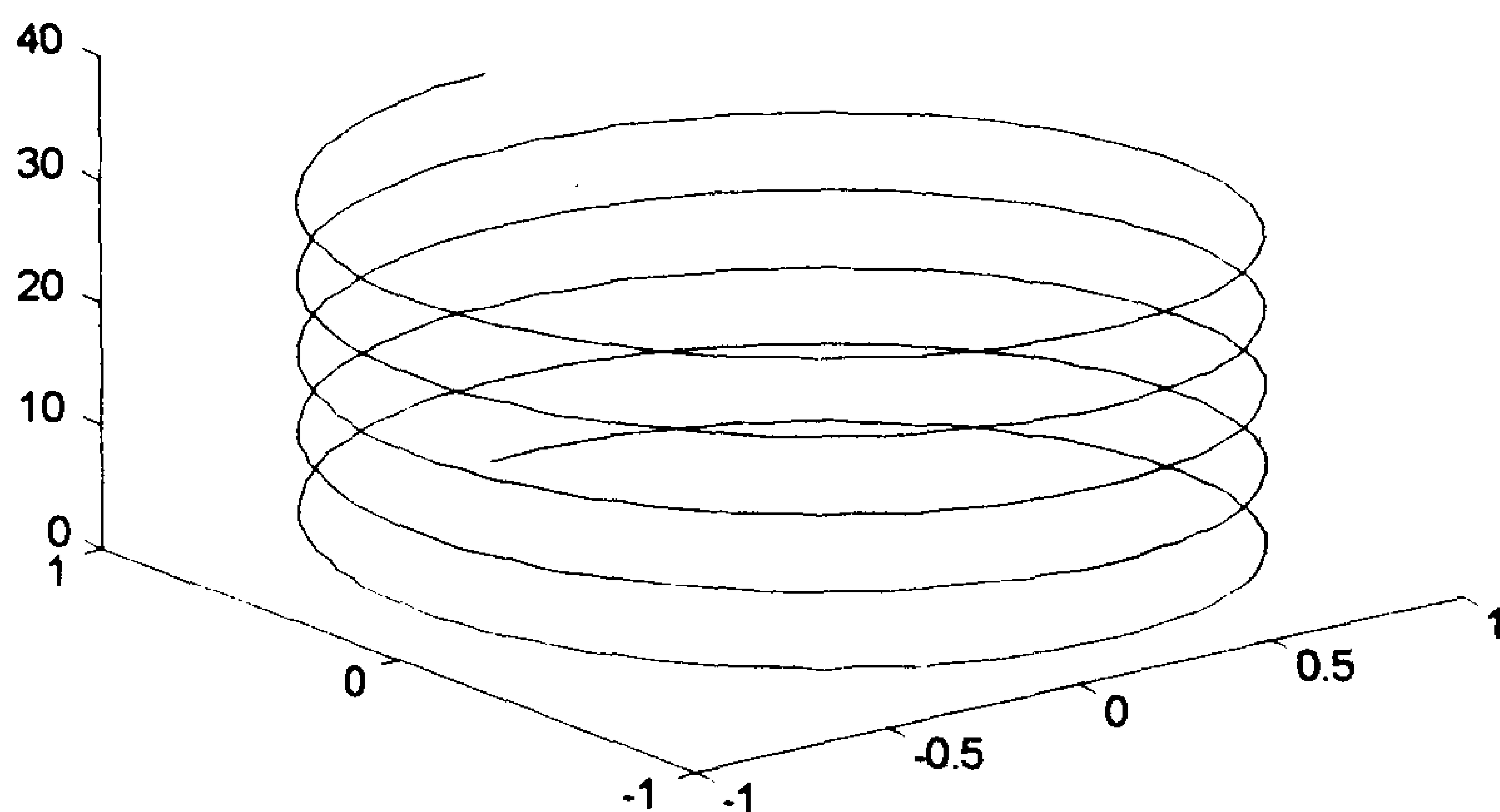


图 2-17　由 plot3() 函数得出的三维曲线

另外，为使得显示的三维图形更漂亮，可以绘制出三维曲面图。在 MATLAB 下允许使用 mesh() 函数来绘制三维表面网格图，这一函数也是 MATLAB 3.x 版本下绘制三维图形的唯一函数，其基本调用格式为

```
mesh(z)
```




其中 z 为一个矩阵，表示一个曲面的 z 坐标。在 MATLAB 4.0 下，该曲面网格绘制函数的调用函数可以写成

```
mesh(x, y, z, c)
```

其中 x 和 y 分别为构成该曲面的 x 和 y 向量，而 c 为颜色矩阵，表示在不同的高度下的颜色范围，如果省略此选项，则 MATLAB 会自动地假定 c=z, 亦即颜色的设定是正比于图形的高度的，这样就可以得出层次分明的三维图形来。

例 2.16　在 x, y 平面内选择一个区域，然后绘制出

$$z = f(x,y) = 3(1-x)^2 e^{-x^2/2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

的三维表面图形。首先可以调用 [x,y] = meshgrid(-3:1/8:3) 函数来设置 x 和 y 平面的网格表示，该函数的调用意义十分明显，即可以产生一个横纵坐标均起始于 -3, 中止于 3，且步距为 1/8 的网格图形。然后由上面的公式计算出曲面的 z 矩阵。最后调用 mesh() 函数来绘制曲面的三维表面网格图形。

```
>>  [x,y] = meshgrid(-3:1/8:3);
>>  z =  3*(1-x).^2.*exp(-(x.^2) - (y+1).^2)- 10*(x/5 - x.^3 - y.^5)...
     .*exp(-x.^2-y.^2)- 1/3*exp(-(x+1).^2 - y.^2);
>>  mesh(x,y,z)
```

可以看出，在计算 z 矩阵时大量地采用了点运算，使得 z 为一个高度矩阵。当然这样画图自动产生的坐标不一定很理想，所以可以调用 axis() 函数来重新设定坐标系，

```
>>  axis([min(min(x)) max(max(x)) min(min(y)) max(max(y)) min(min(z)) max(max(z))])
```

最后得出的曲线如图 2-18 所示。

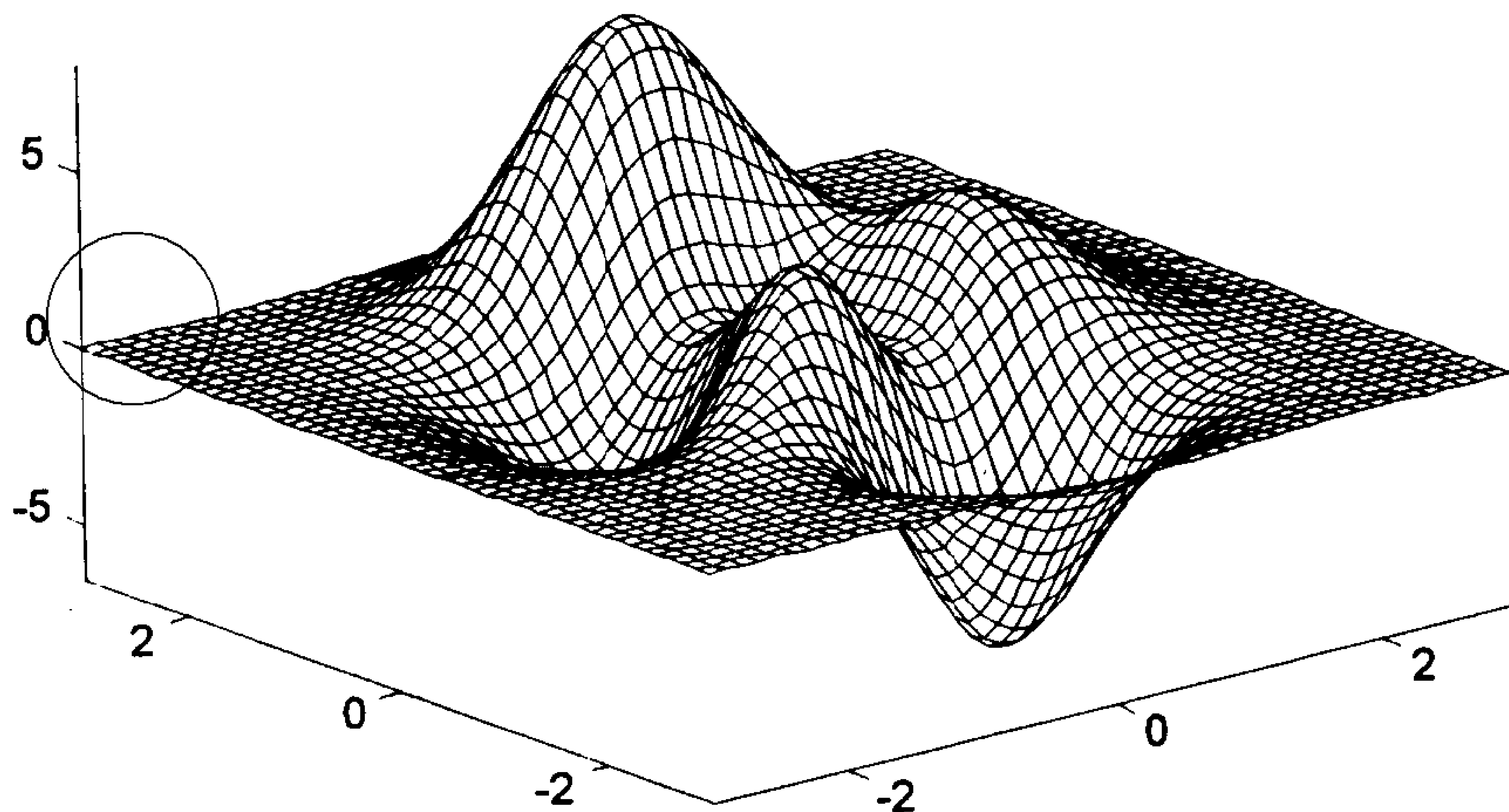


图2-18　由 mesh() 函数得出的三维表面网格图形

由图 2-18 可以看出，在这种默认的状态下隐含的部分都没有绘制出来，如果用户想绘制出隐含的部分，则可以调用 hidden off 命令来进行处理，这时得出的图形如图 2-19 所示，从图中可以

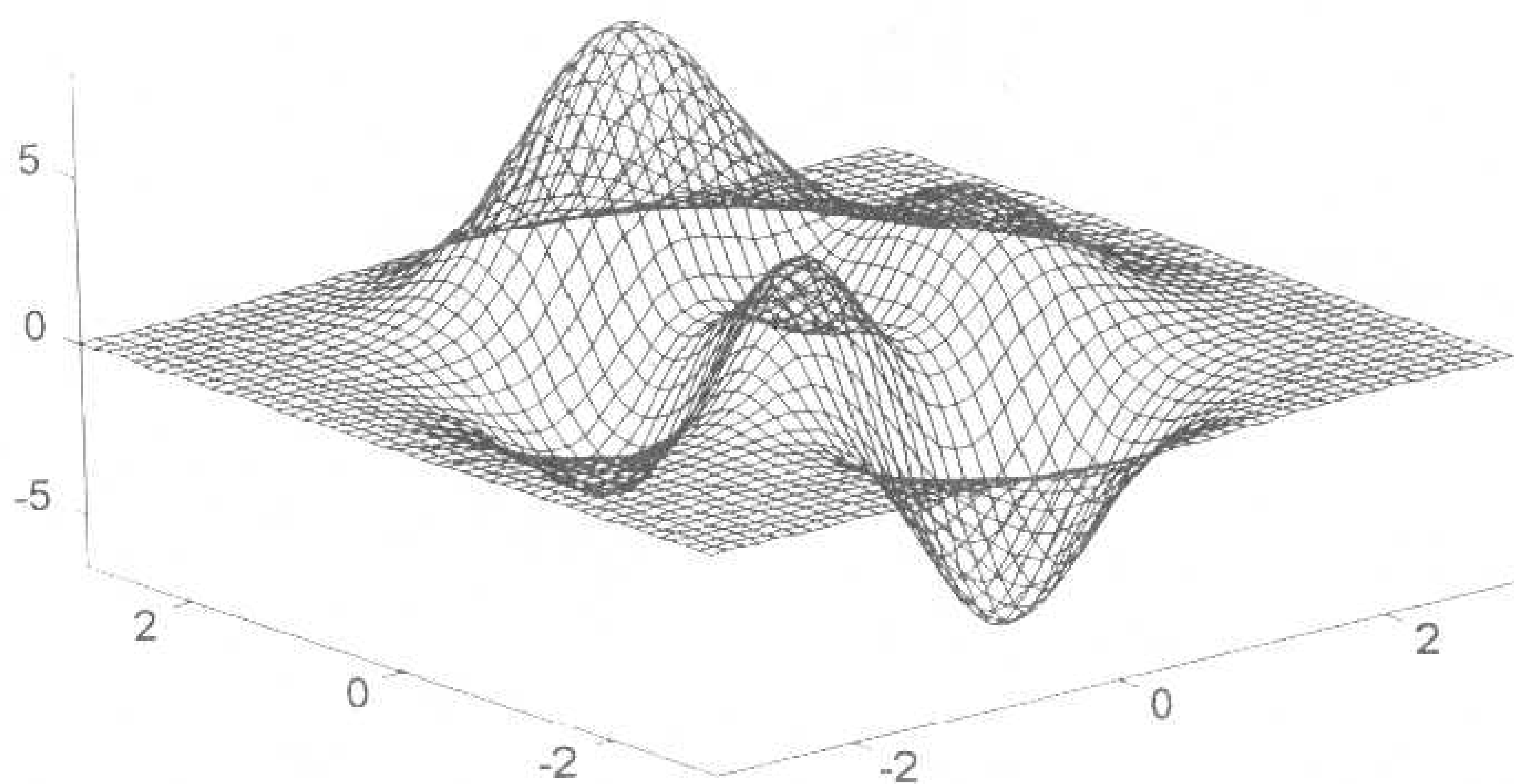


图2-19　hidden off 变换后的网格图

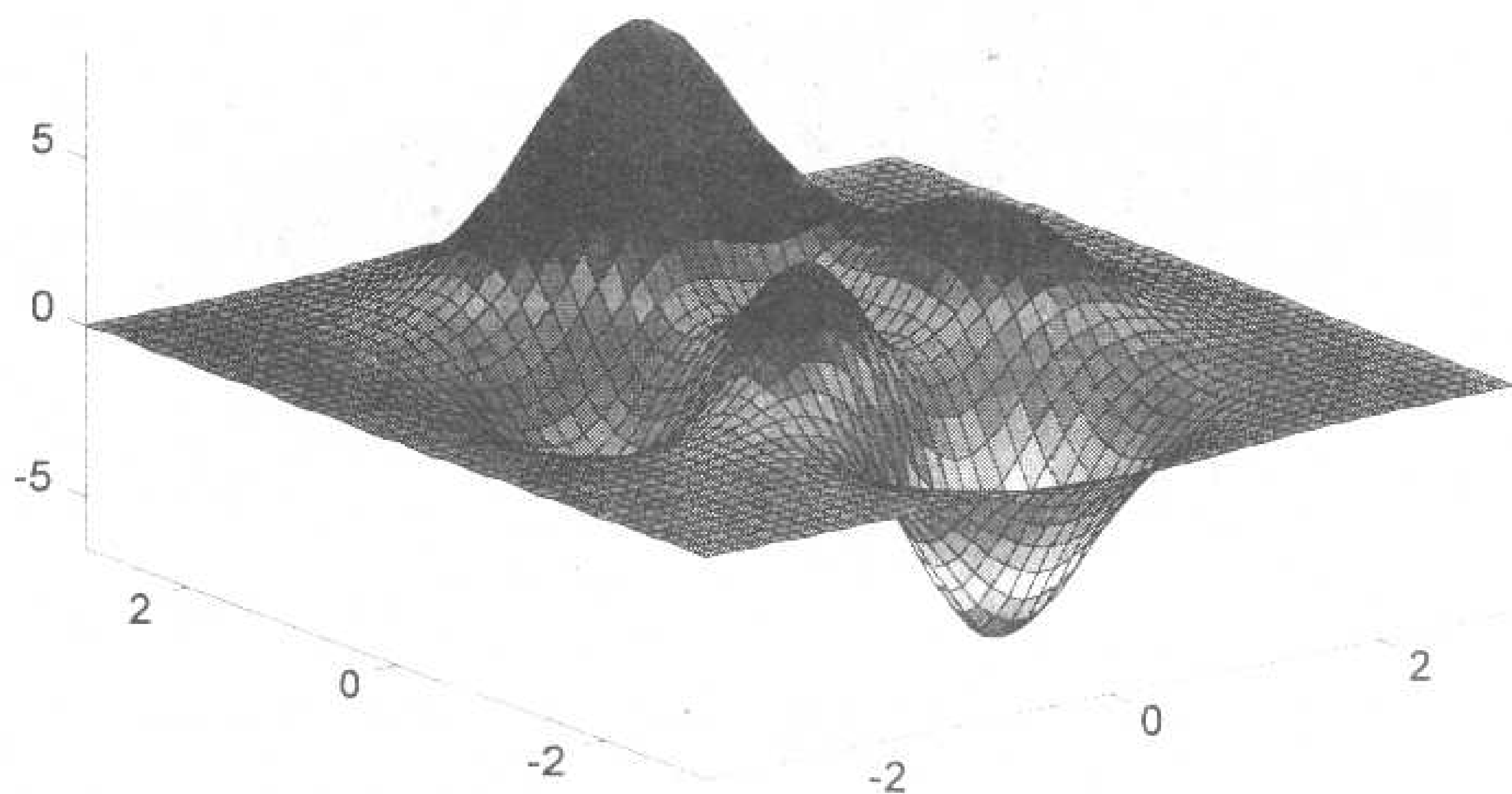


图2-20　由 surf() 函数得出的三维表面图形

如果前面用到的 mesh() 函数用 surf(x, y, z) 函数取代，则绘制出来的表面图形如图 2-20 所示。

如果将前面的 surf() 函数由 surfc() 及 surfl() 取代，则可以分别获得带有等高线 (contour) 的三维图形和带有阴影的三维图形，这两个函数的调用方式和 surf() 函数完全一致，所不同的是绘制出来的图形效果。如果使用这样两个命令，则可以得出如图 2-21 和 2-22 所示的图形。

利用 MATLAB 的绘图功能，用户还可以调用 coutour() 函数容易地绘制出等高线来，亦即如果用户键入 contour(x,y,z) 命令，则会自动绘制出如图 2-23 所示的等高线来，而该等高线的形状将和 2-21 中的一致。

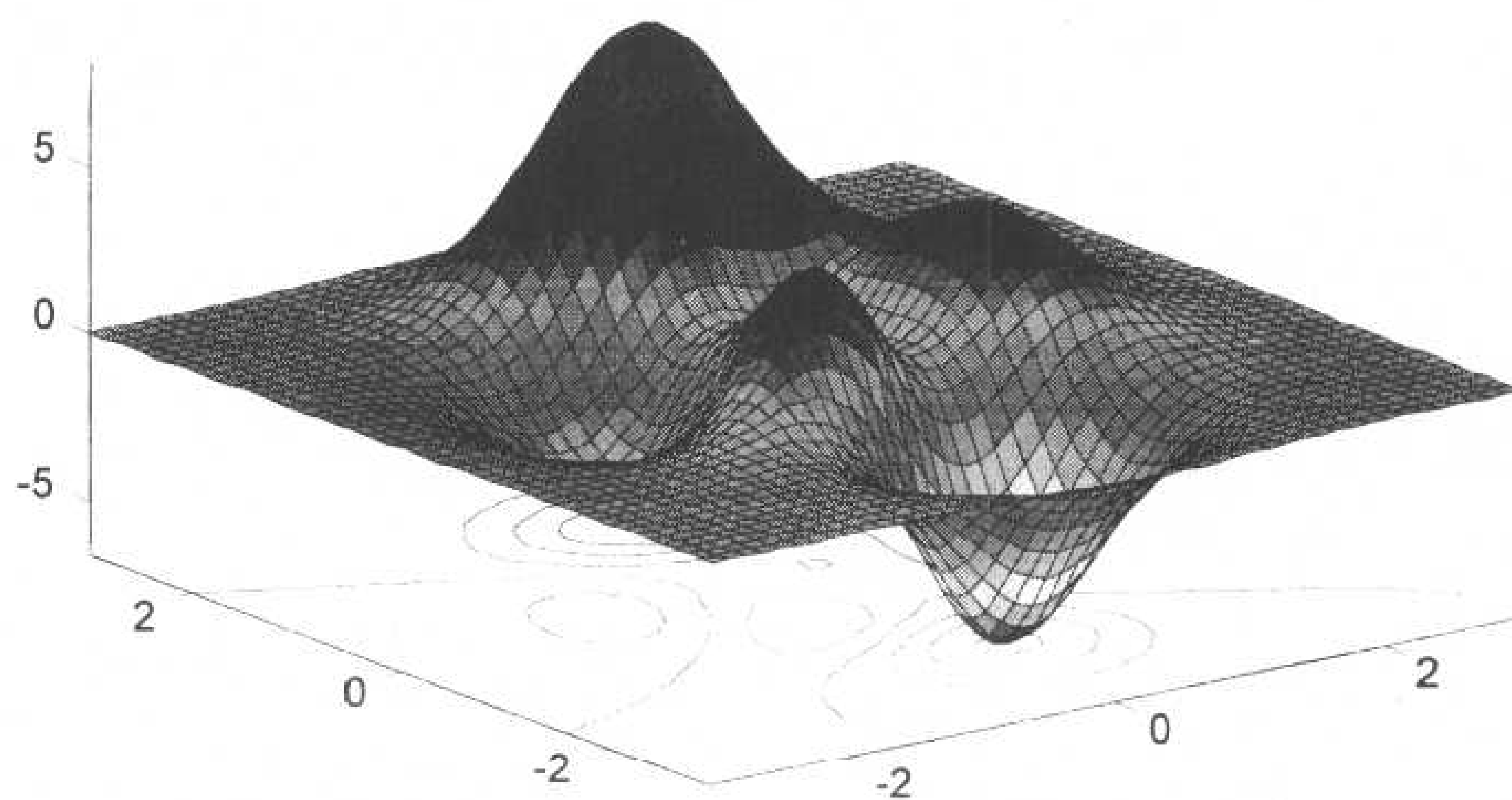


图2-21 由 surfc() 函数得出的带等高线的三维表面图形

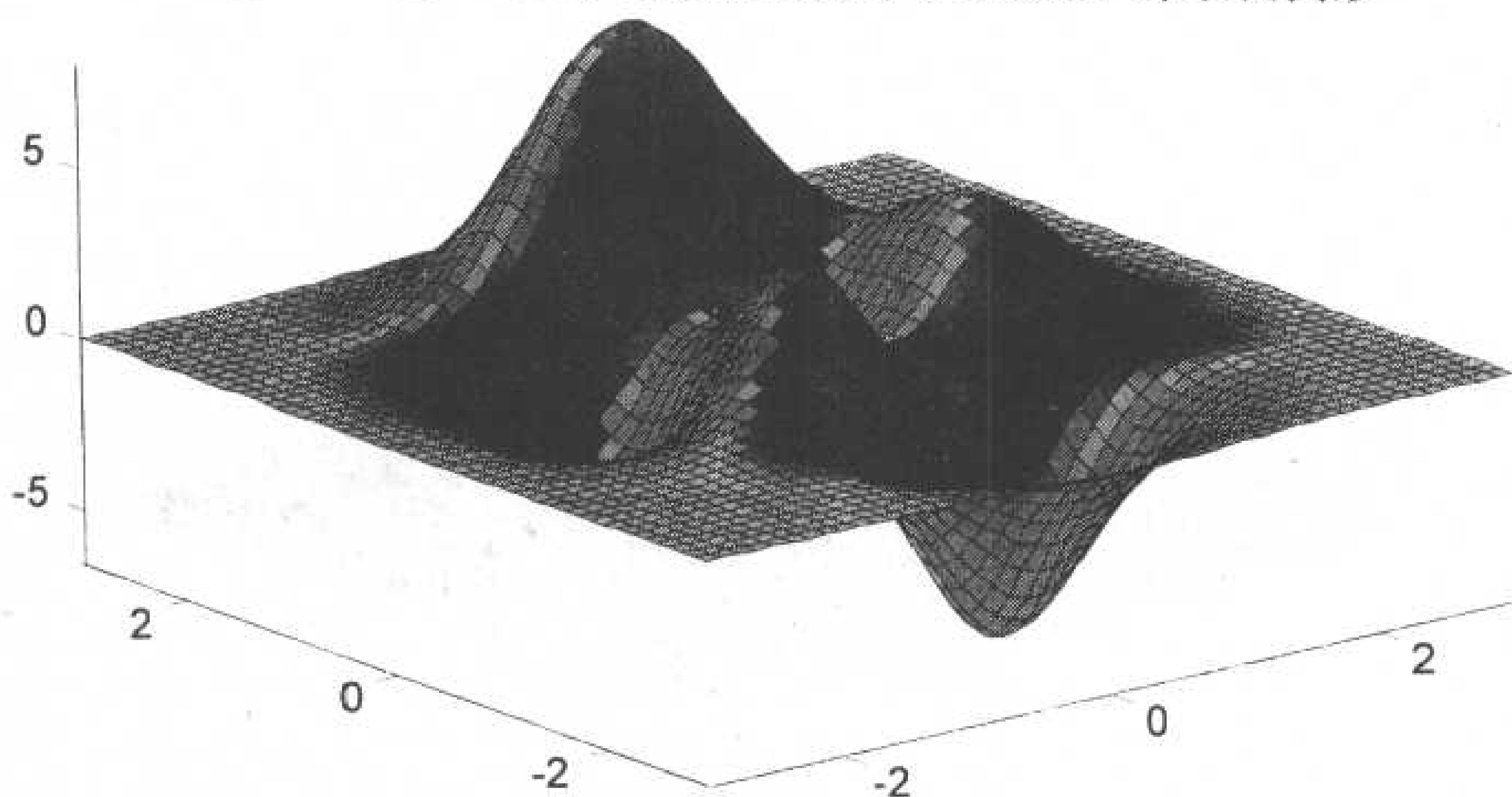


图2-22 由 surfl() 函数得出带有阴影的三维表面图形

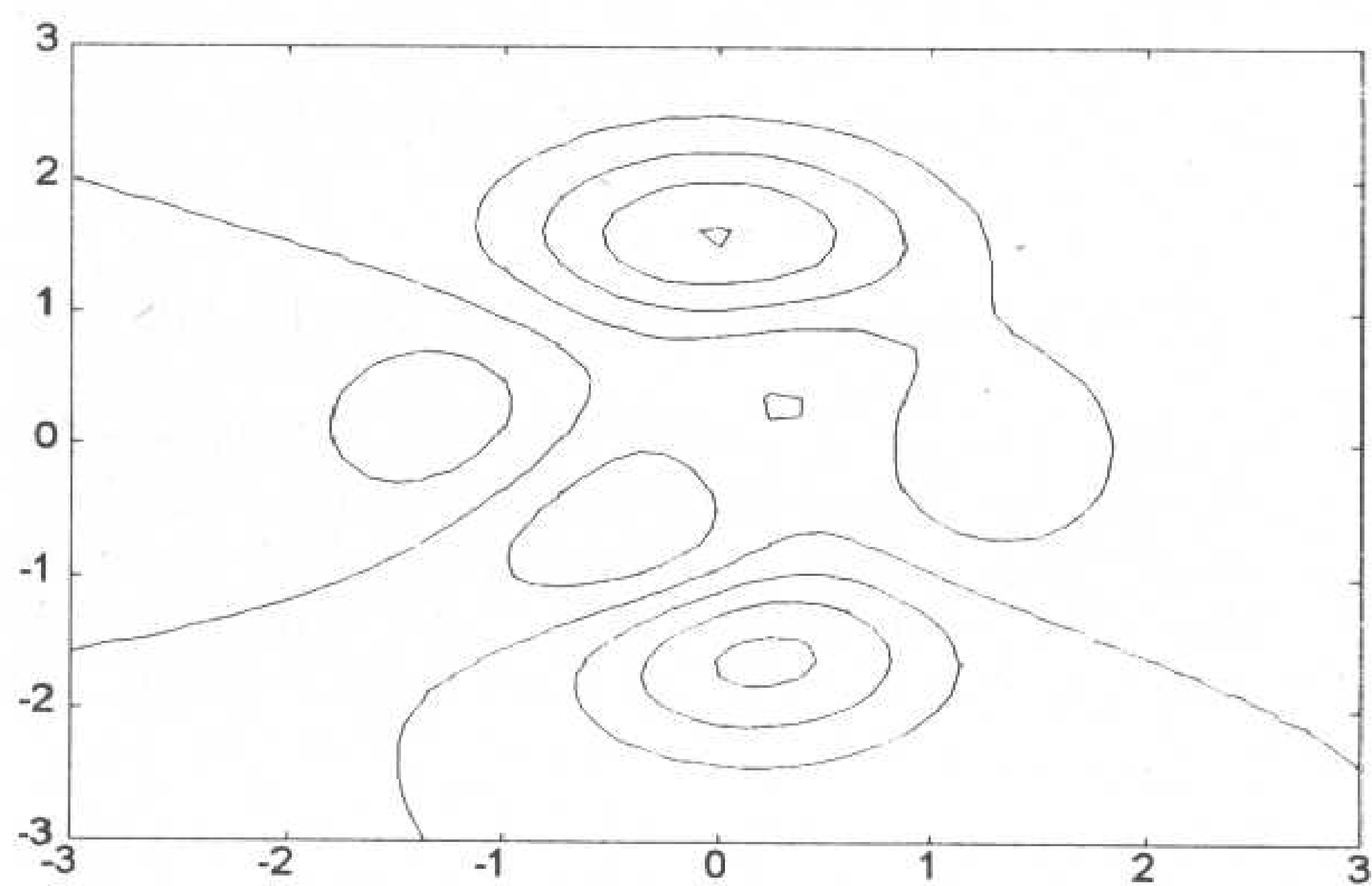


图2-23 由 contour() 函数得出的等高线图形

2.8 MATLAB 编程举例

由前面各节的叙述可以看出，MATLAB 是一个功能极强的高度集成化程序设计语言，它具备一般程序设计语言的基本语句结构，并且它的功能更强，这是诸如 C 或 FORTRAN 语言这样的程序设计语言无法比拟的。由 MATLAB 编写出来的程序结构简单，可读性强，可以说，在掌握了 MATLAB 之后，专门从事控制系统设计和研究的人员没有必要再去弄清 C 语言的细节了。在本节中将给出几个 MATLAB 编程的例子，用户可以从中品味 MATLAB 语言的简洁与直观的编程结构，并从中领略 MATLAB 编程的一些技巧。

例 2.17　当矩阵的阶次很高时，求解矩阵的逆矩阵一般很困难，我们知道，如果一个矩阵可以写成

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{2.8.1}$$

上面的矩阵是一个分块矩阵，这时其逆矩阵可以表示成

$$B = A^{-1} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \tag{2.8.2}$$

其中

$$\begin{aligned} &B_{22} = (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}, \qquad &B_{12} = -A_{11}^{-1}A_{12}B_{22}, \\ &B_{21} = -B_{22}A_{21}A_{11}^{-1}, \qquad &B_{11} = A_{11}^{-1} - B_{12}A_{21}A_{11}^{-1} \end{aligned} \tag{2.8.3}$$

可见这里的整个矩阵求逆就简化成对低阶子矩阵的求逆以及矩阵乘法运算了。然而可以看出，这里需要大量地使用矩阵求逆及乘法运算的调用，所以用一般程序设计语言（如 FORTRAN 及 C 语言）实现起来是相当繁琐的，而用 MATLAB 则可以简洁地编写出如下的程序

```
A  = input('Enter matrix A => ');
[n, m] = size(A);
n1 = input('Enter the size of partition n1=> ');
if (n1>n)
   disp(['n should be smaller than ' int2str(n)])
elseif (n1==n), B = inv(A)
else
   A11 = A(1:n1, 1:n1);  A12=A(1:n1, n1+1:n);
   A21 = A(n1+1:n, 1:n1);  A22=A(n1+1:n, n1+1:n);
   iA11 = inv(A11);
   B22 = inv(A22-A21*iA11*A12); B12 = -iA11*A12*B22;
   B21 = -B22*A21*iA11;  B11 = iA11-B12*A21*iA11;
   B = [B11  B12; B21 B22]
end
```

可见这一程序还是相当简单并直观的。前面几个语句允许用户输入矩阵 A 的内容和要作的分块维数 n1，而后面的语句将按给出的算法来完成分块矩阵的求逆运算。假设整个语句可以写出一个文件，其名称为 test.m，直接运行此程序，则会得出如下的结果

```
>>test
Enter matrix A => [0 0 -1 0 0 0 2 0
```

```
     0 6 0 0 0 -6 0 0; 0 0 0 2 0 0 0 -4
     3 0 0 0 -2 0 1 0; 0 0 6 0 0 0 5 0
     1 0 0 0 -3 0 0 2; 0 4 0 -1 0 0 0 0;
     0 0 1 0 -1 0 0 -2]
Enter the size of partition n1=> 4
B =
  -0.2000        0         0    0.4000    0.0000   -0.2000        0   -0.2000
  -0.0426        0    0.1250   -0.0250    0.0221    0.0750   ,0.2500   -0.1750
  -0.2941        0         0         0    0.1176         0        0         0
  -0.1706        0    0.5000   -0.1000    0.0882    0.3000        0   -0.7000
  -0.1235        0         0    0.1000    0.0294   -0.3000        0   -0.3000
  -0.0426  -0.1667    0.1250   -0.0250    0.0221    0.0750    0.2500   -0.1750
   0.3529        0         0         0    0.0588         0        0         0
  -0.0853        0         0   -0.0500    0.0441    0.1500        0   -0.3500
```

直接调用 inv() 函数，则可以得出如下的结果

```
>>  C = inv(A)
C =
  -0.2000        0         0    0.4000    0.0000   -0.2000        0   -0.2000
  -0.0426        0    0.1250   -0.0250    0.0221    0.0750    0.2500   -0.1750
  -0.2941        0         0         0    0.1176         0        0         0
  -0.1706        0    0.5000   -0.1000    0.0882    0.3000        0   -0.7000
  -0.1235        0         0    0.1000    0.0294   -0.3000        0   -0.3000
  -0.0426  -0.1667    0.1250   -0.0250    0.0221    0.0750    0.2500   -0.1750
   0.3529        0         0         0    0.0588         0        0         0
  -0.0853        0         0   -0.0500    0.0441    0.1500        0   -0.3500
```

可以看出，以上两种算法给出的结果是一致的。进一步分析两个结果，可以调用 norm() 函数来求出两个结果之间误差的最大值为 norm(C-B,'inf') $= 1.1102 \times 10^{-16}$，这样的误差在实际运算中是可以接受的，下面我们可以对结果作进一步误差分析

```
>>  norm(C*A-eye(8))              >>  norm(A*C-eye(8))
ans = 1.4701e-016                 ans = 1.6494e-016
>>  norm(B*A-eye(8))              >>  norm(A*B-eye(8))
ans = 2.0722e-016                 ans = 5.6076e-016
```

从上面的结果可以看出，inv() 函数直接得出的逆矩阵的精度要比间接方法高，所以在使用 MATLAB 时最好采用直接方法。

例 2.18 考虑 Lorenz 模型，其状态方程表示为

$$\begin{cases} \dot{x}_1(t) = -\beta x_1(t) + x_2(t)x_3(t) \\ \dot{x}_2(t) = -\sigma x_2(t) + \sigma x_3(t) \\ \dot{x}_3(t) = -x_2(t)x_1(t) + \rho x_2(t) - x_3(t) \end{cases} \tag{2.8.4}$$

如果选择 $\sigma = 10$, $\rho = 28$, $\beta = 8/3$, 这时 Lorenz 方程的模型在 lorenzeq.m 中给出，其核心部分的清单为



```
function ydot = lorenzeq(t,y)
ydot=[-8/3, 0, y(2); 0, -10, 10; -y(2), 28, -1]*y;
```

有了该方程的数学模型之后，则可以编写下面的程序段(事实上，MATLAB 的演示程序中给出了一个相应的例子，其源程序段可以参见 lorenz.m 文件)，

```
axis([10 40 -20 20 -20 20])
view(3)
hold on
title('Lorenz Attractor')

y0 = [0 0 eps];
tfinal = 100;
y = ode23p('lorenzeq',0,tfinal,y0')
```

在这一程序段中，首先设置了画图的一些参数，诸如对三维图形坐标轴的设定等，并调用了 hold on 命令设置了图形窗口的保护状态，然后在图形窗口中给出了图形的标题，给出这些信息之后，在程序中还定义了该方程的初始条件 y0，并给出了终止仿真的时间 tfinal，最后再调用 ode23p() 函数来对该模型进行仿真，并将结果在图形窗口上输出出来。图形输出是由 ode23p() 函数自动完成的，关于该函数和其它相应的函数及其应用方法后面还将介绍。这段程序最终得出的 Lorenz 曲线如图 2-24 所示。事实上，这样的 Lorenz 方程是一个著名的混沌问题，而这一问题可以通过图示的方式容易地显示出来。

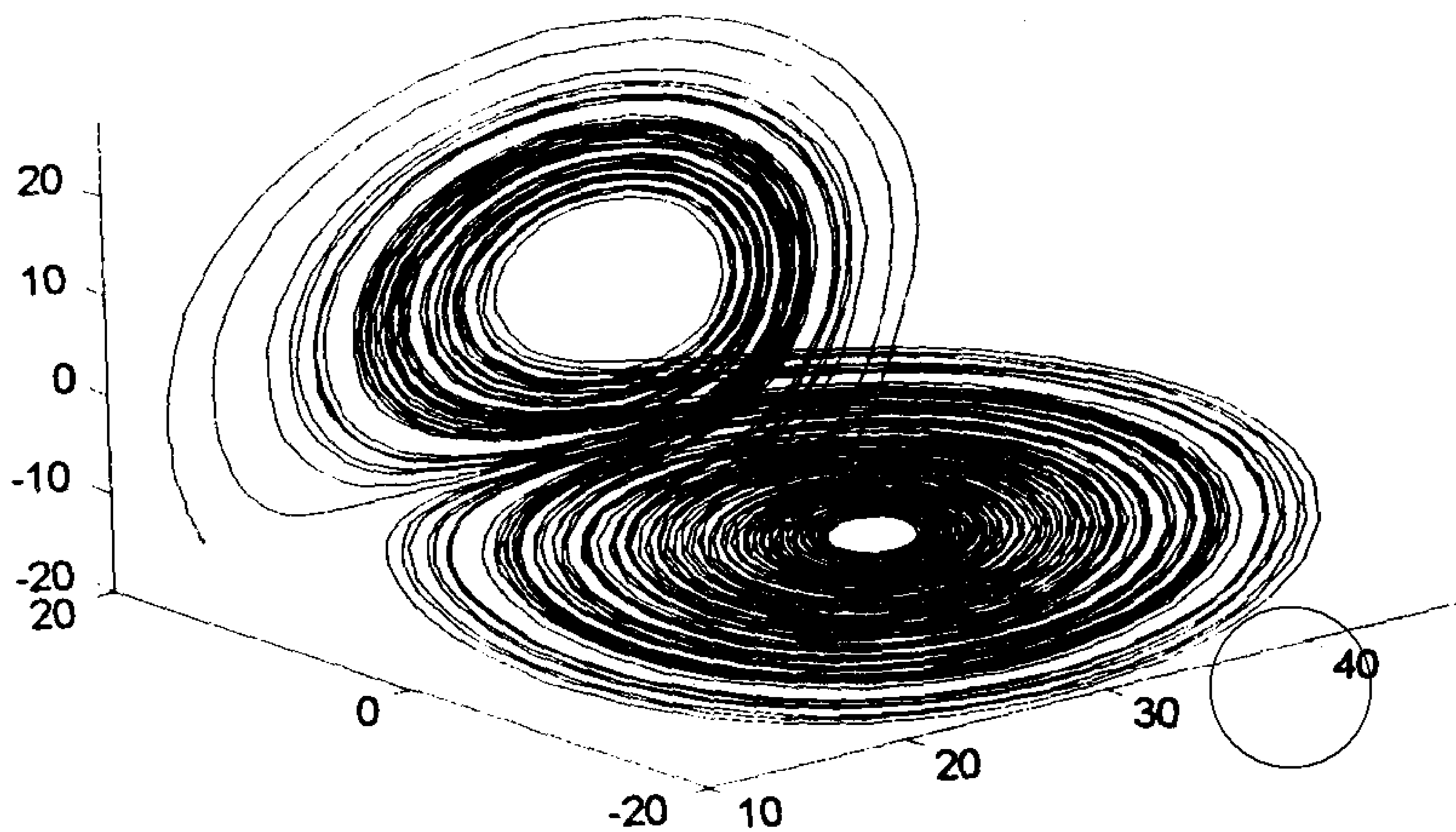


图 2-24　Lorenz 方程的混沌曲线

习　　题

1) 用 MATLAB 可以识别的格式输入下面两个矩阵

$$(a)\quad A=\begin{bmatrix}1&2&3&3\\2&3&5&7\\1&3&5&7\\3&2&3&9\\1&8&9&4\end{bmatrix},\quad (b)\quad B=\begin{bmatrix}1&4&3&6&7&8\\2&3&3&5&5&4\\2&6&5&3&4&2\\1&8&9&5&4&3\end{bmatrix}$$



再求出它们的乘积矩阵 C，并将 C 矩阵的右下角 2×3 子矩阵赋给 D 矩阵，赋值完成之后，调用相应的命令查看 MATLAB 工作空间的占用情况。

2) 解线性方程

$$\begin{bmatrix} 5 & 7 & 6 & 5 & 1 \\ 7 & 10 & 8 & 7 & 2 \\ 6 & 8 & 10 & 9 & 3 \\ 5 & 7 & 9 & 10 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} X = \begin{bmatrix} 24 & 96 \\ 34 & 136 \\ 36 & 144 \\ 35 & 140 \\ 15 & 60 \end{bmatrix}$$

3) 用 MATLAB 语言实现下面的分段函数

$$y = f(x) = \begin{cases} -h, & x > D \\ h/Dx, & |x| \le D \\ -h, & x < -D \end{cases}$$

4) 如果给出下面两个矩阵

$$A = \begin{bmatrix} 4 & 12 & 20 \\ 12 & 45 & 78 \\ 20 & 78 & 136 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

执行下列的矩阵运算命令，并回答有关的问题

(a) `A+5*B` 和 `A-B+I` 分别是多少 (其中 I 为单位矩阵)?

(b) `A.*B` 和 `A*B` 将分别给出什么结果，它们是否相同，为什么？

(c) 得出 `A.^B` 、`A/B` 及 `A\B` 的结果，并分别解释它们的物理意义。

5) 分别用 `for` 和 `while` 循环结构编写程序，求出

$$K = \sum_{i=0}^{63} 2^i = 1 + 2 + 2^2 + 2^3 + \cdots + 2^{62} + 2^{63}$$

并考虑一种避免循环的简洁方法来进行求和。

6) 用循环语句形成一个由 20 个分量的数组，使之元素满足 Fibonacci 规则，即使得数组的第 $k+2$ 满足 $a_{k+2} = a_k + a_{k+1}$, $k = 1, 2, \cdots$，且 $a_1 = 1, a_2 = 1$ 。

7) 选择合适的步距绘制出下面的图形

(a) $\sin\left(\frac{1}{t}\right)$，　其中 $t \in (-1, 1)$

(b) $\sin\tan t - \tan\sin t$，　其中 $t \in (-\pi, \pi)$

8) 对合适的 θ 范围选取分别绘制出下列极坐标图形：

(a) $\rho = 1.0013\theta^2$

(b) $\rho = \cos(7\theta/2)$

(c) $\rho = \sin(\theta)/\theta$

(d) $\rho = 1 - \cos^3(7\theta)$

9) 试以描点法近似绘制图 2-25 曲线

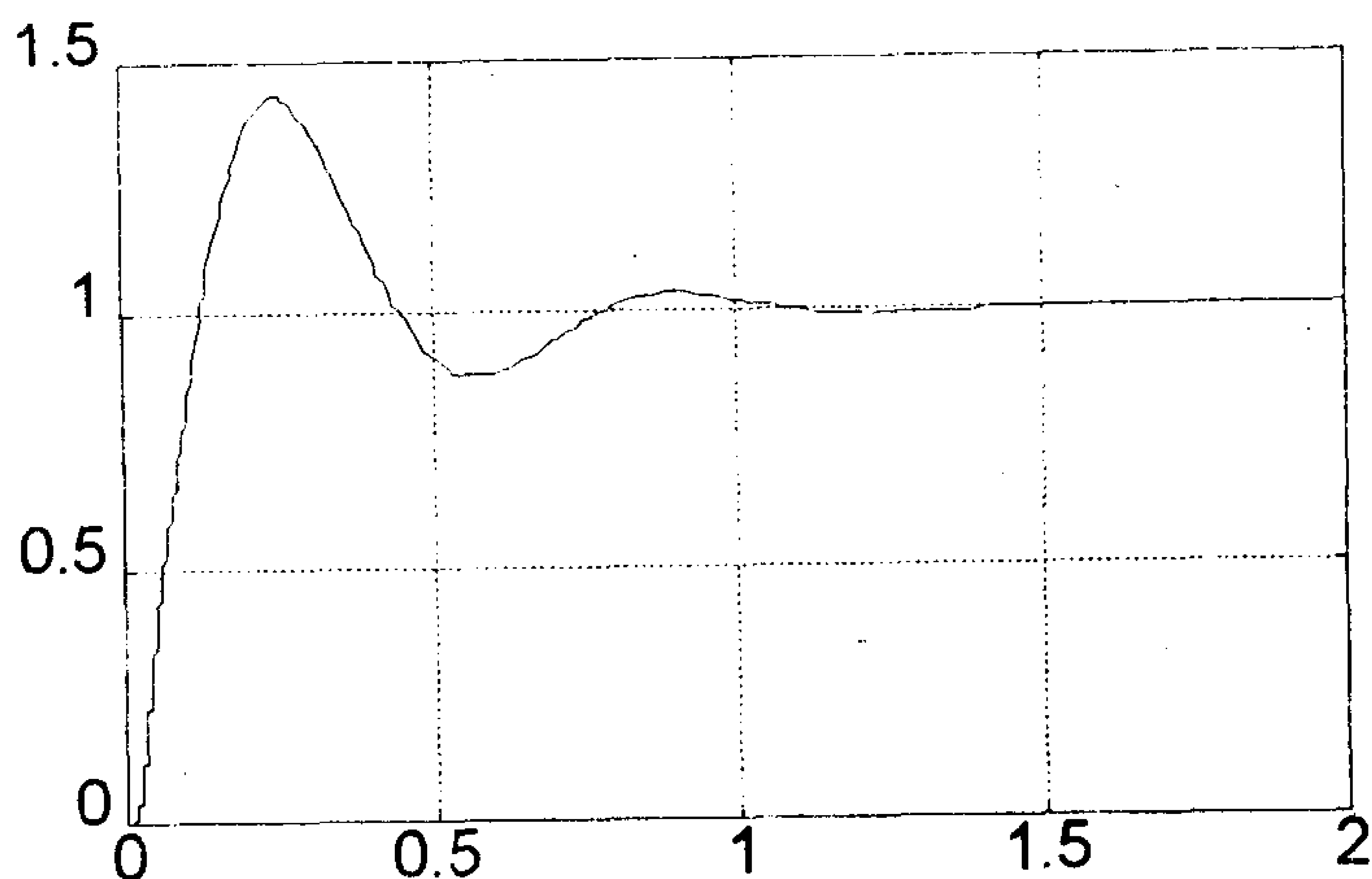


图2-25 习题9)附图

10) 假设一实际问题的解析解由下面分段函数表示[1]

$$p(x_1,x_2)=\begin{cases}0.5457\exp(-0.75x_2^2-3.75x_1^2-1.5x_1), & x_1+x_2>1\\ 0.7575\exp(-x_2^2-6x_1^2), & -1<x_1+x_2\leq 1\\ 0.5457\exp(-0.75x_2^2-3.75x_1^2+1.5x_1), & x_1+x_2\leq -1\end{cases}$$

试以三维曲面的形式来表示这一函数。

11) 体会其它绘图命令及效果，假设用户有下面的绘图向量 `x=0:.1:2*pi` 及 `y=sin(x)`，试利用下面各个函数 `polar()`，`bar()`，`stem()`，`stairs()`，`errorbar()`，`hist()`，`rose()`，`compass()`，`feather()`，`comet()` 的调用得出不同的图形，并体会每条命令及函数调用的方法和意义。

12) 用 MATLAB 产生一个有 10000 个元素的正态分布随机向量，然后绘制出产生的随机变量的概率分布，并与其理论值得出的曲线进行比较。如果结果不令人满意，则可以增大随机元素个数，再进行比较。

(提示：MATLAB 提供了 `randn()` 函数来生成随机元素矩阵，如果用 `x=randn(1,10000)` 则可以得出由 10000 个元素组成的行向量，对这一向量求出直方图数据可以调用 `hist()` 函数来完成，然后再对结果进行规范化。)

13) 利用本章中介绍的分块矩阵求逆算法计算出矩阵

$$A=\begin{bmatrix}2&1&4&-2&-1&-4\\3&0&-1&-3&0&1\\2&3&4&-2&-3&-4\\4&2&8&6&3&12\\6&0&-2&9&0&-3\\4&6&8&6&9&12\end{bmatrix}$$

的逆矩阵，并比较分块维数 n1 分别取 1, 2, 3, 4, 5 时求逆精度的情况。

14) 由矩阵理论可知，如果一个矩阵 M 可以写成 $M=A+BCB^T$，其中 A,B,C 为相应阶数的矩阵，则 M 矩阵的逆矩阵可以由下面的算法求出

$$M^{-1}=\left(A+BCB^T\right)^{-1}=A^{-1}-A^{-1}B\left(C^{-1}+B^TA^{-1}B\right)^{-1}B^TA^{-1}$$

试根据上面的算法用 MATLAB 语句编写一个函数对矩阵 M 进行求逆，并通过一个小例子来检验该程序。

15) 下面给出了一个迭代模型

$$\begin{cases} x_{k+1} = 1 + y_k - 1.4x_k^2 \\ y_{k+1} = 0.3x_k \end{cases}$$

取迭代初值为 $x_0 = 0, y_0 = 0$，并进行 30000 次迭代求出一组 x 和 y 向量，然后用在所有的 $x(k)$ 和 $y(k)$ 坐标处点亮一个点 (注意不要连线)，最后绘制出所需的图形。(提示：这样绘制出的图形又称为 Henon 引力线图，它将迭代出来的随机点吸引到一起，最后得出貌似连贯的引力线图)。

参 考 文 献

[1] Atherton D P, Xue, D. The analysis of feedback systems with piecewise linear nonlinearities when subjected to Gaussian inputs. In: Kozin F, Ono T (eds.). Control systems, topics on theory and application. Tokyo: Mita Press, 1991, pp23-38

[2] Dongarra J J, Bunsh J R, Molor C B. LINPACK user's guide. Philadelphia: Society of Industrial and Applied Mathematics (SIAM), 1979

[3] Forsythe G E, Malcolm M A, Moler C B. Computer methods for mathematical computations. Englewood Cliffs: Prentice-Hall, 1977

[4] Forsythe G E, Moler C B. Computer solution of linear algebraic systems. Englewood Cliffs: Prentice-Hall, 1967

[5] Garbow B S. Boyle J M, Dongarra J J, Moler C B. Matrix eigensystem routines - EISPACK guide extension, Lecture notes in computer sciences. Vol. 51, New York: Springer-Verlag, 1977

[6] Kahaner D, Moler C B, Nash S. Numerical Methods and Software. Englewood Cliffs: Prentice Hall, 1989

[7] MathWorks. MATLAB™, High-performance numeric computation and visualization software—External interface guide, 1993

[8] MathWorks. MATLAB™, High-performance numeric computation and visualization software, Users' guide, 1993

[9] Smith B T, Boyle J M, Dongarra J J *et al.* Matrix eigensystem routines - EISPACK guide, Lecture notes in computer sciences. Vol. 6 (Second edition). New York: Springer-Verlag, 1976

第3章 数值线性代数方法及MATLAB实现

虽然线性代数中的数值计算一般都可以直接用MATLAB求出，但作者认为如果想进一步研究CACSD技术，最好能了解其中一些典型问题的解法，所以在本章中将系统地介绍数值线性代数的常用概念、算法及MATLAB实现。

一般的数值算法都可以用来对矩阵进行计算和操作，但这些算法的本身是否可靠，则需要用一些标准的测试问题来检验，这些特殊的矩阵的选择必须是客观的，文献[5]中列出了很多可以用来测试矩阵运算方法的特殊矩阵。读者可以自己利用该书中给出的例子来测试MATLAB环境是否可靠。

在本章中首先将介绍一些特殊矩阵的概念及MATLAB实现，然后将逐步介绍矩阵特征参数的求取、矩阵的各种分析方法、矩阵的特征值及求逆等一般的数值线性代数问题，本章中还将叙述稀疏矩阵的MATLAB实现、矩阵的非线性函数求取及包括数据处理、数值积分、非线性方程求解与最优化，以及微分方程初值问题在内的其它数值分析方法及MATLAB实现，可见MATLAB为求解各种各样的控制问题提供了重要的手段。

3.1 特殊矩阵的实现

在开始介绍矩阵运算之前，有必要先介绍一些特殊矩阵的定义及MATLAB实现，这里要遇到的很多运算当然可以由MATLAB直接做到。

- **零矩阵和单位矩阵**: 在实际矩阵运算中，引入零矩阵和单位矩阵的定义是很有用的。在一般矩阵理论中，把所有元素都为零的矩阵定义成零矩阵，其MATLAB实现为

```
A=zeros(n,m)
```

这里n和m分别为矩阵A的行数和列数。如果使用了命令`A=zeros(n)`，则将产生一个$n\times n$零矩阵。如果B是一个给定的矩阵，则在MATLAB早期版本中允许使用`A=zeros(B)`来定义一个和B阵同样大小的零矩阵，但这样的定义有时会出现问题，例如当A矩阵是一个正整数时，MATLAB无法断定想产生一个常数还是产生一个方阵，所以这样定义有时会产生混乱，为避免这种易混淆的现象发生，在MATLAB 4.0及以上版本中规范地将之定义为`A=zeros(size(B))`。

单位矩阵的定义为一个方阵(行数等于列数)，其主对角线元素均为1，而其它元素全部为0。在本书中单位矩阵的数学表示记作I，其MATLAB实现为`A=eye(n)`。MATLAB还允许将这一单位矩阵扩展为`A=eye(m,n)`，这样就可以产生一个$m\times n$



的矩阵，其主对角线的元素为 1，其余全部为 0。相应地，如果有一个给定矩阵 B，和它维数相同的单位矩阵 A 也可以由如下定义 A=eye(size(B)) 来产生。

- **随机元素矩阵**: 顾名思义，随机元素矩阵的各个元素是随机产生的，如果矩阵的随机元素满足 $[0,1]$ 区间上的均匀分布，则可以由 MATLAB 函数 rand() 来生成，该函数通常的调用格式为

```
A=rand(n,m)
```

其中 n 和 m 分别为要产生随机矩阵的行数和列数。如果要产生一个和 A 矩阵相同阶次的随机元素矩阵，则仍然可以使用 B=rand(size(A)) 来进行赋值。这里所用到的随机函数发生器的种子值为 MATLAB 默认的，如果想改变随机函数的种子值，则可以通过下面的调用方式来完成

```
rand('seed',s)
```

其中 s 是一个数值，用户可以通过 s 参数的设置来改变种子值，如果想设置回默认值，则可以给出命令 rand('seed',0) 。例如如果在不同的种子下连续两次调用 rand() 函数，则获得的随机元素矩阵的内容是不相同的，而将种子值设置成默认值，则将和第一次调用的结果相同

```
>>  rand(1,6)
ans =  0.2190    0.0470    0.6789    0.6793    0.9347    0.3835
>>  rand(1,6)
ans =  0.5194    0.8310    0.0346    0.0535    0.5297    0.6711
>>  rand('seed',0);
>>  rand(1,6)
ans =  0.2190    0.0470    0.6789    0.6793    0.9347    0.3835
```

早期的 MATLAB 版本中 rand() 函数还允许选择其它的随机分布形式，但在 MATLAB 4.0 中如果想产生满足标准正态分布的随机元素矩阵，则应该调用 randn() 函数来完成，该函数的标准调用格式仍为

```
A=randn(n,m)
```

如果想改变随机函数发生器的种子值，则仍可以采用 randn('seed',s) 来设定。

在 MATLAB 中采用了伪随机数发生器的方式来产生随机数据，所以若其分布方式和随机数种子确定下来之后，伪随机数序列就唯一地确定了，故它可以保证输出可重复的随机数序列。

- **对角矩阵**: 对角矩阵是一种特殊的矩阵，这种矩阵的主对角线元素可以为非零或零元素，而非对角线元素的值均为 0 。对角矩阵的数学描述方法为 $\mathrm{diag}(\alpha_1,\alpha_2,\cdots,\alpha_n)$，



其中对角矩阵的矩阵表示为

$$\mathrm{diag}(\alpha_1,\alpha_2,\cdots,\alpha_n)=\begin{bmatrix}\alpha_1 & & & \\ & \alpha_2 & & \\ & & \ddots & \\ & & & \alpha_n\end{bmatrix} \tag{3.1.1}$$

如果用 MATLAB 提供的方法建立一个向量 $V_d=[\alpha_1,\alpha_2,\cdots,\alpha_n]$，则对角矩阵可以用 MATLAB 函数 diag() 来建立。例如给定一个向量 V=[1 2 3 4]，则 W = diag(V) 将得出一个对角矩阵

```
>>  V=[1 2 3 4];  diag(V)
ans =  1     0     0     0
       0     2     0     0
       0     0     3     0
       0     0     0     4
```

其对角线上各个元素由向量 V 给出。对角矩阵行列式的值为全部对角元素的乘积，即 $\prod_{i=1}^{n}\alpha_i$。如果矩阵 A 为一个方阵，则调用 V=diag(A) 将提取出 A 矩阵的对角元素来构成向量 V，而不管 A 矩阵的非对角元素是何值。

- Hilbert **及逆** Hilbert **矩阵**: Hilbert 矩阵是一类特殊矩阵，它的第 (i,j) 元素的值满足 $h_{i,j}=1/(i+j-1)$，这时一个 $n\times n$ 阶的 Hilbert 矩阵可以写成

$$H=\begin{bmatrix}1 & 1/2 & 1/3 & \cdots & 1/n\\ 1/2 & 1/3 & 1/4 & \cdots & 1/(n+1)\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1/n & 1/(n+1) & 1/(n+2) & \cdots & 1/(2n-1)\end{bmatrix} \tag{3.1.2}$$

产生 Hilbert 矩阵的 MATLAB 函数 hilb() 的调用方式为

```
H=hilb(n)
```

其中 n 为要产生的矩阵阶次。事实上，Hilbert 矩阵也可以由下面的 MATLAB 语句直接产生

```
for i=1: n
   for j=1:n
      H(i,j) = 1/(i+j-1);
end,  end
```

高阶 Hilbert 矩阵一般为坏条件的矩阵，所以直接对之求逆一般往往会引出浮点溢出的现象。MATLAB 提供了直接求取逆 Hilbert 矩阵的算法及函数，其调用方法为

```
B=invhilb(n)
```




其中 n 为方阵的维数。然而 Hilbert 矩阵毕竟是坏条件矩阵，所以一般情况下如果 n 比较大，例如 $n \geq 11$，则不能保证 $BA = AB = I$，这样就不适于用普通矩阵的求逆方法，而只能采取 invhilb() 函数来求其逆矩阵了。

- **伴随矩阵** (companion matrix)：在控制理论中有一类比较重要的矩阵，称作伴随矩阵。假设有一个多项式

$$P(s) = p_0 s^n + p_1 s^{n-1} + p_2 s^{n-2} + \cdots + p_{n-1}s + p_n \tag{3.1.3}$$

且有 $p_0 \neq 0$, 对此多项式进行首一化处理，亦即将多项式的各项均除以首项的系数 p_0，则可以获得如下的新多项式 $P_0(s)$ 满足

$$P_0(s) = s^n + a_1 s^{n-1} + a_2 s^{n-2} + \cdots + a_{n-1}s + a_n \tag{3.1.4}$$

式中 $a_i = p_i/p_0,\ i = 1, 2, \cdots, n$ 。这时可以写出一个伴随矩阵 [1]

$$A_c = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \cdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tag{3.1.5}$$

生成伴随矩阵的 MATLAB 函数为 compan()，其调用方法为

```
B=compan(p)
```

其中 p 为一个多项式向量。

例 3.1 例如有一个向量 P=[1,2,3,4,5]，则可以通过下面的命令构成一个伴随矩阵

```
>> P = [1, 2, 3, 4, 5];
>> A = compan(P)
A =  -2    -3    -4    -5
      1     0     0     0
      0     1     0     0
      0     0     1     0
```

- **Hankel 矩阵**: 假设有一个序列 C, 其各个元素为 $\{c_1, c_2, \cdots, c_n, \cdots\}$, 这时可以写出一个如下的矩阵，这一矩阵的第 (i, j) 元素满足 $h_{i,j} = c_{i+j-1},\ i, j = 1, 2, \cdots, n$ 。这样可以构造一个矩阵

$$H = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \\ c_2 & c_3 & \cdots & c_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ c_n & c_{n+1} & \cdots & c_{2n-1} \end{bmatrix} \tag{3.1.6}$$

[1] 注意：为适合 MATLAB 的控制系统工具箱中定义的系统矩阵格式，这里所用的伴随矩阵表示形式可能与一些控制理论书籍中有差异。



这样的矩阵称为 Hankel 矩阵。如果 $n \to \infty$, 则可以构造无穷型 Hankel 矩阵。Hankel 矩阵是对称矩阵，且其反对角线上的元素相同。在 MATLAB 下，如果已知一个向量 C, 则可以由 hankel() 函数来构造出一个 Hankel 矩阵。例如取 C=[1,2,3]，则可以构造一个 3 阶的 Hankel 矩阵

```
>>  H=hankel(C)
H = 1     2     3
    2     3     0
    3     0     0
```

可见，如果直接用单个向量来生成 Hankel 矩阵，则会自动地将该向量的各个元素填写到矩阵的第一列中去，然后利用其反对角线元素的值相等这一特点写出其它元素，而主反对角线下的各个元素均设置为 0 。MATLAB 还提供了该函数的另外一种调用方法，给定两个向量 C 和 R，如果用 H=hankel(C, R) 来生成 H，则首先将 H 矩阵的第一列的各个元素定义为 C 向量，而最后一列各个元素定义为 R，这样就可以依照 Hankel 矩阵反对角线上元素相等这一特性来写出相应的 Hankel 矩阵来。例如，若 C=[1,2,3]; R=[8,9,10]; 则将得出下面的结果

```
>>  C=[1,2,3]; R=[8,9,10];
>>  H = hankel(C, R)
H =  1     2     3     8
     2     3     8     9
     3     8     9    10
```

- **Vandermonde 矩阵**: 假设有一个序列 C, 其各个元素满足 $\{c_1, c_2, \cdots, c_n\}$, 这时可以写出一个如下的矩阵，这一矩阵的第 (i,j) 元素满足 $v_{i,j} = c_i^{n-j}$, $i,j = 1,2,\cdots,n$ 。这样可以构成一个矩阵

$$V = \begin{bmatrix} c_1^{n-1} & c_1^{n-2} & \cdots & c_1 & 1 \\ c_2^{n-1} & c_2^{n-2} & \cdots & c_2 & 1 \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ c_n^{n-1} & c_n^{n-2} & \cdots & c_n & 1 \end{bmatrix} \tag{3.1.7}$$

这种矩阵称作 Vandermonde 矩阵。如果已知一个向量 C, 则可以由 MATLAB 提供的 V= vander(C) 函数来构造一个 Vandermonde 矩阵，例如有向量 C=[1, 2, 3, 4, 5]，则可以得出该向量对应的 Vandermonde 矩阵为

```
>>  C=[1, 2, 3, 4, 5];
>>  V=vander(C)
V =   1      1      1      1      1
     16      8      4      2      1
     81     27      9      3      1
    256     64     16      4      1
    625    125     25      5      1
```




3.2 矩阵的特征参数运算

MATLAB 提供了大量的矩阵特征参数求取函数，在这里首先叙述各个矩阵特征参数的意义和求解方法，然后介绍各个参数的 MATLAB 求解方法。

- **矩阵的行列式**: 矩阵 $A=\{a_{ij}\}$ 的行列式定义为

$$D = |A| = \det(A) = \sum (-1)^k a_{1k_1} a_{2k_2} \cdots a_{nk_n} \tag{3.2.1}$$

式中 $k_1, k_2, \cdots, k_n$ 是将序列 $1, 2, \cdots, n$ 的元素交换 k 次所得出的一个序列，每个这样的序列称为一个置换 (permutation)，而 Σ 表示对 $k_1, k_2, \cdots, k_n$ 取遍 $1, 2, \cdots, n$ 的所有的排列的求和。

计算矩阵的行列式有多种算法，在 MATLAB 中采用的方法为首先对原矩阵 A 进行三角分解 (又称为 LU 分解，后面将介绍)，将该矩阵分解成一个上三角矩阵 U 和一个下三角矩阵 L 的积，即 $A=LU$ ，这样首先可以求出 L 矩阵的行列式，注意在这一矩阵中只有一种排列方式且其行列式的值 s 为 1 或 -1 。同样因为 U 阵为上三角矩阵，所以其行列式的值为该矩阵主对角线元素之积，则 A 矩阵行列式为 $\det(A) = s\prod_{i=1}^{n} U_{ii}$ 。

MATLAB 下提供了内部函数 det()，该函数可以直接求出矩阵的行列式．det() 函数的调用格式为

```
det(A)
```

例 3.2　假设矩阵 A 如下给出

$$A = \begin{bmatrix} 10 & 5 & 4 & 3 & 2 & 1 \\ -1 & 10 & 5 & 4 & 3 & 2 \\ -2 & -1 & 10 & 5 & 4 & 3 \\ -3 & -2 & -1 & 10 & 5 & 4 \\ -4 & -3 & -2 & -1 & 10 & 5 \\ -5 & -4 & -3 & -2 & -1 & 10 \end{bmatrix}$$

则由后面一系列函数的调用可以容易地求出该矩阵的一些特征参数

```
>>  A = [10, 5, 4, 3, 2, 1; -1, 10, 5, 4, 3, 2;
       -2, -1, 10, 5, 4, 3;  -3, -2, -1, 10, 5, 4;
       -4, -3, -2, -1, 10, 5; -5, -4, -3, -2, -1, 10];
>>  det(A)
ans =       1402768
```

为统一起见，在本节的其它部分中将直接对同样的原矩阵 A 求取其它矩阵参数。

- **矩阵的迹**: 假设一个方阵为 $A=\{a_{ij}\}, i,j=1,2,\cdots,n$ ，则矩阵 A 的迹定义为

$$\operatorname{tr}(A) = \sum_{i=1}^{n} a_{ii} \tag{3.2.2}$$



亦即矩阵的迹为该矩阵对角线上各个元素之和。由代数理论可知，矩阵的迹和该矩阵的特征值之和是相同的。在 MATLAB 也提供了求取矩阵迹的函数 trace()，其调用方法可以自然地写成

```
trace(A)
```

这样用户只需将矩阵名填写到该函数中即可。

考虑前面给出的矩阵 A, 直接调用 MATLAB 函数 trace() 则可以求取该矩阵的迹

```
>> trace(A)
ans =     60
```

- **矩阵的秩**: 若矩阵所有的列向量中共有 r_c 个线性无关，则称矩阵的列秩为 r_c, 如果 $r_c = m$, 则称 A 为列满秩矩阵，相应地，若矩阵 A 的行向量中有 r_r 个是线性无关的，则称矩阵 A 的行秩为 r_r，如果 $r_r = n$, 则称 A 为行满秩矩阵。可以证明，矩阵的行秩和列秩是相等的，记

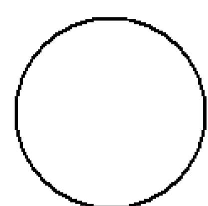
$$\operatorname{rank}\{A\} = r_c = r_r \tag{3.2.3}$$

这时矩阵的秩为 $\operatorname{rank}\{A\}$。矩阵的秩也表示该矩阵中行列式不等于 0 的子式的最大阶次，所谓子式，即为从原矩阵中任取 k 行及 k 列所构成的子矩阵。

矩阵求秩的算法也是多种多样的，其区别是有的算法是稳定的，而有的算法可能因矩阵的条件数变化不是很稳定，MATLAB 中采用的算法是基于矩阵的奇异值分解的算法 [1]: 首先对矩阵作奇异值分解，得出矩阵 A 的 n 个奇异值 σ_i, $i = 1, 2, \cdots, n$，在这 n 个奇异值中找出大于给定误差限 ε 的个数 r，这时 r 就可以认为是 A 矩阵的秩。

MATLAB 提供了一个内部函数 rank() 来用数值方法求取一个已知矩阵的秩，其调用方法为

```
k=rank(A, tol)
```



其中 A 为要求秩的矩阵，而 tol=ε 为判 0 用的误差限，一般可以取默认值 eps，这样调用格式就可以简化为 k=rank(A)，这里的判 0 用误差限就取作机器中的默认值 eps。如果 eps 取得不合适，则求出的数值秩可能和原矩阵的秩不同，所以在使用数值秩时应当引起注意。

考虑前面给出的矩阵 A, 直接调用 MATLAB 函数 rank(A) 则可以求取该矩阵的秩

```
>>  rank(A)
ans =      6
```




- **矩阵的范数**: 矩阵的范数是对矩阵的一种测度，在介绍矩阵的范数之前，首先要介绍向量范数的基本概念。

  如果对线性空间中的一个向量 x 存在一个函数 $\rho(x)$ 满足下面三个条件

  1) $\rho(x) \leq 0$ 且 $\rho(x)=0$ 的充要条件是 $x=0$;
  2) $\rho(ax)=|a|\rho(x)$, a 为任意标量；
  3) 对向量 x 和 y 有 $\rho(x+y)\leq\rho(x)+\rho(y)$

  则称 $\rho(x)$ 为 x 向量的范数。范数的形式是多种多样的，可以证明，下面给出的一族式子都满足上述的三个条件

$$\|x\|_p=\left(\sum_{i=1}^{n}|x_i|^p\right)^{1/p},\ p=1,2,\cdots,\text{且}\ \|x\|_\infty=\max_{1\leq i\leq n}|x_i| \tag{3.2.4}$$

  这里用到了向量范数的记号 $\|x\|_p$ 。

  矩阵的范数定义比向量的稍复杂一些，其完全的定义如下：对于任意的非零向量 x, 矩阵 A 的范数为

$$\|A\|=\sup_{x\neq 0}\frac{\|Ax\|}{\|x\|} \tag{3.2.5}$$

  和向量的范数一样，对矩阵来说也有常用的范数定义方法

$$\|A\|_1=\max_{1\leq j\leq n}\sum_{i=1}^{n}|a_{ij}|,\ \|A\|_2=\sqrt{s_{\max}\{A^TA\}},\ \|A\|_\infty=\max_{1\leq i\leq n}\sum_{j=1}^{n}|a_{ij}| \tag{3.2.6}$$

  其中 $s\{X\}$ 为 X 矩阵的特征值，而 $s_{\max}\{A^TA\}$ 即为 A 矩阵的最大奇异值的平方。换句话说， $\|A\|_2$ 为 A 矩阵的最大奇异值。

  MATLAB 提供了求取矩阵范数的函数 norm()，允许求各种意义下的矩阵范数。该函数的调用格式为

```
N = norm(A, 选项)
```

  其中允许的选项如表 3-1 所示。

考虑前面给出的矩阵 A, 直接调用 MATLAB 函数 norm(A) 并选择不同的选项参数，则可以求取该矩阵的各种范数

```
>>  norm(A)
ans =    18.2911
>>  norm(A,2)
ans =    18.2911
>>  norm(A,1)
```




表3-1 矩阵范数函数的选项表

选　项	意　义　及　算　法
无	矩阵的最大奇异值，即 $\|A\|_2$
2	与默认调用方式相同，亦为即 $\|A\|_2$
1	矩阵的 1- 范数，即 $\|A\|_1$
Inf 或'inf'	矩阵的无穷范数，即 $\|A\|_\infty$
'fro'	矩阵的 F- 范数，即 $\|A\|_F=\sqrt{\sum(A^TA)_{ii}}$
数值 P	对向量可取任何整数，而对矩阵只可取 1，2，inf 或'fro'
-inf	只可用于向量， $\|A\|_{-\infty}=\min(\sum a_i)$

```
ans =     25
>>  norm(A,Inf)
ans =     25
>>   norm(A,'fro')
ans =    30.4959
```

这里有两点值得注意：首先 norm(A) 和 norm(A,2) 应该给出同样的结果，因为它们都表示 $\|A\|_2$，另外由于例子中给出的 A 矩阵为对称矩阵，即 $A^T=A$，所以对称的 1- 范数和无穷范数的值是一致的，一般情况下 $\|A\|_1\neq\|A\|_\infty$ 。

- **矩阵的特征多项式、特征方程与特征根**: 构造一个矩阵 $sI-A$, 并求出该矩阵的行列式，则可以得出一个多项式 $C(s)$

$$C(s)=\det(sI-A)=s^n+c_1s^{n-1}+\cdots+c_{n-1}s+c_n \tag{3.2.7}$$

这样的多项式 $C(s)$ 称为矩阵 A 的特征多项式，其中系数 $c_i, i=1,2,\cdots,n$ 称为矩阵的特征多项式系数。

MATLAB 提供了求取矩阵特征多项式系数的函数 poly()，其调用格式为

```
C = poly(A)
```

其中 A 为给定的矩阵，而返回的 C 为一个行向量，其各个分量为矩阵 A 的降幂排列的特征多项式系数。

考虑例 3.2 中给出的矩阵 A, 直接调用 MATLAB 函数 poly(A) 当然可以求出该矩阵的特征多项式系数为

```
>>  format long
>>  B = poly(A)
```




```
B =   1.0e+006 *
   0.00000100000000  -0.00006000000000   0.00160500000000  -0.02316400000000
   0.18692400000000  -0.79684800000000   1.40276800000000
```

由于矩阵 A 的特征多项式系数相差太悬殊，所以调用了 format long 以显示更多位的有效数字。由前面得出的结果可见，此矩阵的特征多项式可以写成

$$P(s) = s^6 - 60s^5 + 1605s^4 - 23164s^3 + 186924s^2 - 796848s + 1402768$$

事实上，$P(s)$ 多项式即为原矩阵特征多项式的理论解，下面我们将分析由 poly() 函数调用结果产生的相对误差

```
>> P = [1, -60, 1605, -23164, 186924, -796848, 1402768];
>> norm((P-B)./P)
ans =  2.1636e-015
```

由上面的结果可见，MATLAB 提供的 poly() 函数在计算矩阵的特征多项式系数是会产生微小的误差。

MATLAB 在计算矩阵特征多项式时程序的核心是下面的程序段

```
z=eig(A); c=zeros(n+1,1); c(1)=1;
for j=1:n
   c(2:j+1) = c(2:j+1)-z(j)*c(1:j);
end
```

可见，在这段程序中调用了 eig() 函数来求特征值，而 eig() 函数的计算是很复杂的，产生些小误差在所难免，而这一误差最终又被传递到其它函数的计算中去，所以在使用时应引起注意。

在实际应用中当然还有其它简单的方法可以求出矩阵的特征多项式系数，如下面给出的 Fadeev-Fadeeva 递推算法也可以求出矩阵的特征多项式

$$\begin{cases} c_k = -\dfrac{1}{k}\mathrm{tr}(AR_k), & k = 1, 2, \cdots, n \\ R_1 = I,\ R_k = AR_{k-1} + c_{k-1}I, & k = 2, \cdots, n \end{cases} \tag{3.2.8}$$

该算法首先给出一个单位阵 R_1，并将之赋给 R，然后对每个 k 的值分别求出特征多项式参数，并更新 R 矩阵，最终得出矩阵的特征多项式系数。该算法可以直接由下面的 MATLAB 语句来编写一个 poly1() 函数来实现

```
function c=poly1(A)
n=length(A); R0 = eye(n); R=R0; c=[1 zeros(1,n)];
for k=1:n
   c(k+1)=-1/k*trace(A*R);  R=A*R+c(k+1)*R0;
end
```

同样考虑前面的例子，如果调用上面的程序段，则可以得出如下的结果

```
>>  c=poly1(A)
c =    1   -60    1605    -23164    186924    -796848    1402768
>>  norm((P-c)./P)
ans =     0
```

可见这样得出的特征多项式的系数均为整数 (精确解)，此外由于这里给出的算法运算起来比较简单，不需要求取矩阵的特征值，所以运算量也比 poly() 的小，甚至可以得出精确的解来。

令特征多项式等于零所构成的方程称为该矩阵的特征方程，而特征方程的根称为该矩阵的特征根。特征根当然可以由后面将要介绍的矩阵特征值算法直接求出，如果获得了矩阵的特征方程，则矩阵的特征根还可以通过求解多项式方程而求出，这可以调用 MATLAB 函数 roots() 而直接获得，该函数的调用格式为

```
V = roots(P)
```

其中 P 为特征多项式的系数向量，而 V 为特征方程式的解，即原矩阵的特征根。

考虑前面给出的矩阵 A 及特征多项式 $B(s)$, 直接调用 MATLAB 函数 roots(B) 则可以求出该矩阵的特征根

```
>>  roots(B)
ans =  14.4641 +11.1962i
       14.4641 -11.1962i
        8.0000 + 3.0000i
        8.0000 - 3.0000i
        7.5359 + 0.8038i
        7.5359 - 0.8038i
```

例 3.3 考虑文献 [2] 中引用的一个例子：求解多项式方程

$$P(s)=\prod_{i=1}^{20}(s-i)=s^{20}-210s^{19}+20615s^{18}+\cdots+20!$$

的根。利用传统的多项式求根方法可以得出下面的结果 [2]

```
 1.000000000          10.095266145+0.643500904*i
 2.000000000          10.095266145-0.643500904*i
 3.000000000          11.793633881+1.652329728*i
 4.000000000          11.793633881-1.652329728*i
 4.999999928          13.992358137+2.518830070*i
 6.000006944          13.992358137-2.518830070*i
 6.999697234          16.730737466+2.812624894*i
 8.007267603          16.730737466-2.812624894*i
 8.917250294          19.502439400+1.940330347*i
20.846908101          19.502439400-1.940330347*i
```

可见这样的结果有很大的误差，且产生了一些为数不小的虚数部分。为什么在数值算法下无法得出精确解呢？在数值算法下总是采用有限的位数来表示数值，而不能像 Mathematica 这样的符号



运算软件那样保存全部数值，例如它分解出第一个根之后，由于最多保留 15 位有效数字的原因，剩余的多项式系数已经作了某种舍入，使得它和理论值就出现了差异，有时尽管这样的差异是极其微小的，也会产生较大的传递误差。

在 MATLAB 下采用了基于矩阵特征值的求解方法，从而在一定程度上能改进多项式求根的精度，对同样的问题调用 MATLAB 的 roots() 函数可以得出如下的结果

```
>> p=1; for i=1:20, p=conv(p,[1,-i]); end
>> format long; roots(p); [ans(1:10) ans(11:20)]
ans =  19.99983701975324   9.99820277269920
       19.00159255256143   9.00029223959138
       17.99273696893571   7.99998166121775
       17.01942332745671   6.99999779393985
       15.96273966167644   6.00000035281905
       15.04831606986690   5.00000002998320
       13.95323172299049   3.99999999112210
       13.03439570803671   3.00000000055041
       11.98235993811902   1.99999999999236
       11.00689218868810   0.99999999999997
```

从上面得出的结果可以看出，虽然这样的方法避免了得出带有虚部的根，但精度仍然不是很高。分析原始问题可以看出，该多项式方程系数向量中的系数相差过大，例如 s^{20} 的系数为 1，而 s^0 的系数为 2.4329×10^{18}. 对这样的特殊问题产生一些误差是在所难免的。求取直到 20 阶的方程尚有一定的困难，若想像 Mathematica 那样求出直至 50 阶方程的解析解简直是不可能的，所以在求解这样的问题时像 MATLAB 这样的数值软件是有一定局限性的，但这并不过多地妨碍 MATLAB 的实际应用，因为一般情况下用它来求解的问题并不总处于这样的极端情况。即便出现了这样的极端情况，至少它可以保持到 0.001 级的精度。

- **多项式及多项式矩阵的求值**：在第 2 章中介绍过多项式的求值可以由 polyval() 函数直接完成，对于多项式矩阵来说， MATLAB 提供了 polyvalm() 函数，该函数的调用格式为

```
B=polyvalm(aa, A)
```

其中 aa 为多项式系数降幂排列构成的向量，即 aa=$[a_1,a_2,\cdots,a_n,a_{n+1}]$，而 A 为一个给定矩阵，这时返回的矩阵 B 为下面的矩阵多项式的值

$$B=a_1A^n+a_2A^{n-1}+\cdots+a_nA+a_{n+1}I \tag{3.2.9}$$

例 3.4　Hamilton-Cailey 定理是矩阵理论中的一个比较重要的定理，它的内容为：若矩阵 A 的特征多项式为

$$\det(sI-A)=a_1s^n+a_2s^{n-1}+\cdots+a_ns+a_{n+1} \tag{3.2.10}$$

则有

$$a_1A^n+a_2A^{n-1}+\cdots+a_nA+a_{n+1}I=0 \tag{3.2.11}$$

假设有一个矩阵 A 为 A=[1,2,3; 4,5,6; 7,8,0]，可以由下面的 MATLAB 语句来验证 Hamilton-Cailey 定理

```
>> A=[1,2,3; 4,5,6; 7,8,0];
>> aa=poly(A);
>> B=polyvalm(aa, A);
>> norm(B)
ans = 1.6040e-012
```

由于使用的 poly() 函数会产生一定的误差，所以得出的 B 矩阵并不是很精确。如果将上面的语句中 aa = poly(A) 用 poly1() 代替，则得出的 B 矩阵完全等于 0，这样就对该矩阵验证了 Hamilton-Cailey 定理。

3.3 矩阵的相似变换与分解

3.3.1 矩阵的相似变换与正交变换

假设有一个 $n\times n$ 的方阵 A，并存在一个和它同阶的 (亦即 $n\times n$ 的) 非奇异矩阵 T，则可以对 A 矩阵进行如下的变换

$$\widehat{A}=T^{-1}AT \tag{3.3.1}$$

这种变换称为 A 的相似变换 (similarity transform)。可以证明，变换后的矩阵 $\widehat{A}$ 的特征值和原矩阵 A 是一致的，亦即相似变换并不改变原矩阵的特征结构。

对于一类特殊的相似变换矩阵 T 来说，如果它本身满足 $T^{-1}=T^{*}$，其中 T^{*} 为 T 的 Hermit 共轭转置矩阵，则称 T 为正交矩阵，并将之记为 $Q=T$，可见正交矩阵满足下面的条件

$$Q^{*}Q=I,\ \ \text{且}\ \ QQ^{*}=I \tag{3.3.2}$$

其中 I 为 $n\times n$ 的单位阵。

正交矩阵中还有一类特殊的形式，如果 A 矩阵不是满秩矩阵，且 Z 阵为正交矩阵，亦即它满足 $Z^{*}Z=I$，则如果矩阵 Z 可以使得

$$AZ=0 \tag{3.3.3}$$

则称 Z 矩阵为化零空间 (null space)。

MATLAB 中提供了求取正交矩阵 orth() 和化零矩阵 null() 的函数，这两个矩阵的调用方式分别为

```
Q = orth(A),     Z = null(A)
```

其中前一个函数由矩阵 A 构成一个正交基 (orthogonal basis)，亦即它的各个列可以张成 A 矩阵的各列同样的空间，且 Q 的各列为正交的。调用 MATLAB 提供的 null() 函数可以获得前面提及的化零空间，如果 A 为满秩矩阵，则不存在这样的矩阵 Z，这时 null() 函数将返回一个空的矩阵。

例 3.5 重新考虑例 3.2 中给出的矩阵 A，分别调用前面介绍的两个函数，并进行相关的操作，则可以获得下面的结果。



```
>>  Q=orth(A)
Q =  0.2329   -0.8216    0.1591   -0.1349   -0.4707    0.0756
     0.2911    0.0600    0.2447    0.9124   -0.1333    0.0389
     0.5822    0.1201    0.2361   -0.2110    0.1011   -0.7322
    -0.0582    0.2457    0.8692   -0.2507   -0.0372    0.3414
     0.6987    0.2730   -0.3050   -0.1955   -0.0901    0.5459
    -0.1747    0.4149   -0.1028   -0.0610   -0.8608   -0.2052
>>  I = eye(size(A));
>>  norm(Q*Q'-I)
ans =  4.0979e-016
>>  norm(Q'*Q-I)
ans =  4.6029e-016
>> Z = null(A)
Z =     []
```

可见，通过这样的函数则可以建立起来一个正交矩阵 Q, 且这一矩阵满足式 (3.3.2) 中规定的条件。后一个函数返回一个空的矩阵，这说明不存在一个矩阵可以使得 A 转化到零矩阵。

再考虑一个非满秩矩阵 A

$$A=\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&2.5&0\\0&0&0&0\end{bmatrix}$$

```
>>  A = [1, 0, 0, 0; 0, 0, 0, 0; 0, 0, 2.5, 0; 0, 0, 0, 0];
>>  Z = null(A)
Z =     0      0
        1      0
        0      0
        0      1
>>  Z' * Z
ans =   1      0
        0      1
>>  A*Z
ans =   0      0
        0      0
        0      0
        0      0
```

可见，这样得出的化零空间矩阵是一个正交矩阵，且它为 A 矩阵的基，并使得 $AZ=0$。

3.3.2 矩阵的三角分解及 Cholesky 分解

矩阵的三角分解又称为 LU 分解，它的目的是将一个矩阵分解成一个下三角矩阵 L 和一个上三角矩阵 U 的乘积，亦即可以写成 $A=LU$，其中 L 和 U 矩阵分别可以写成

$$L=\begin{bmatrix}1&&&\\l_{21}&1&&\\\vdots&\vdots&\ddots&\\l_{n1}&l_{n2}&\cdots&1\end{bmatrix},\quad U=\begin{bmatrix}u_{11}&u_{12}&\cdots&u_{1n}\\&u_{22}&\cdots&u_{2n}\\&&\ddots&\vdots\\&&&u_{nn}\end{bmatrix} \tag{3.3.4}$$

由这两个矩阵可以简单地写出一个矩阵 A^F ，其中

$$A^F = L + U - I = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & u_{nn} \end{bmatrix} \tag{3.3.5}$$

这样产生的矩阵与原来的 A 矩阵的关系可以写成

$$\begin{array}{llll} a_{11} = u_{11}, & a_{12} = u_{12}, & \cdots & a_{1n} = u_{1n} \\ a_{21} = l_{21}u_{11}, & a_{22} = l_{21}u_{12} + u_{22}, & \cdots & u_{2n} = l_{21}u_{1n} + u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} = l_{n1}u_{11}, & a_{n2} = l_{n1}u_{12} + l_{n2}u_{22}, & \cdots & a_{nn} = \sum_{k=1}^{n-1} l_{nk}u_{kn} + u_{nn} \end{array} \tag{3.3.6}$$

由上式可以立即得出求取 l_{ij} 和 u_{ij} 的递推计算公式

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}}{u_{jj}}, \quad (j < i), \quad 及\ u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, \quad (j \geq i) \tag{3.3.7}$$

该公式的递推初值为

$$u_{1i} = a_{1i}, \quad i = 1, 2, \cdots, n \tag{3.3.8}$$

注意，在上述的算法中并未对主元素进行任何选取，因此该算法并不一定数值稳定，在 MATLAB 下也给出了矩阵的 LU 分解函数 `lu()` ，该函数的调用格式为

```
[L, U] = lu(A)
```

其中 L, U 分别为变换后的下三角和上三角矩阵，在 MATLAB 的 `lu()` 函数中考虑了主元素选取的问题，所以该函数一般会给出可靠的结果。由该函数得出的下三角矩阵 L 并不一定是一个真正的下三角矩阵，因为选取它可能进行了一些元素行的交换，这样主对角线的元素可能不是 1 ，而在矩阵 L 内存在一个唯一的如式 (3.2.1) 中定义的置换，其各个元素的值均是 1 。如果想获得有关换行信息，则可以由下面的格式调用该函数

```
[L, U, P] = lu(A)
```

式中 P 为排列规则矩阵，而这时 L 和 U 分别为真正的下三角和上三角矩阵。但使用这一调用规则时一定要注意，若 P 不为单位阵时，得出的 L 和 U 阵不满足 $A = LU$，而满足 $A = P^{-1}LU$ 。

考虑例 3.2 中给出的矩阵 A 及特征多项式 $B(s)$，直接调用 MATLAB 函数 `lu()`，则可以求出该矩阵的 LU 分解



```
>>  [L, U, P]=lu(A)
L =   1.0000         0         0         0         0         0
     -0.1000    1.0000         0         0         0         0
     -0.2000    0.0000    1.0000         0         0         0
     -0.3000   -0.0476    0.0423    1.0000         0         0
     -0.4000   -0.0952    0.0106    0.0506    1.0000         0
     -0.5000   -0.1429   -0.0212    0.0214    0.0400    1.0000
U =  10.0000    5.0000    4.0000    3.0000    2.0000    1.0000
           0   10.5000    5.4000    4.3000    3.2000    2.1000
           0         0   10.8000    5.6000    4.4000    3.2000
           0         0         0   10.8677    5.5661    4.2646
           0         0         0         0   10.7764    5.3502
           0         0         0         0         0   10.5624
P =    1     0     0     0     0     0
       0     1     0     0     0     0
       0     0     1     0     0     0
       0     0     0     1     0     0
       0     0     0     0     1     0
       0     0     0     0     0     1
```

当然这里的排列规则矩阵 P 为一个单位阵，这样对称的 L 矩阵确实是一个下三角矩阵。

例 3.6　再考虑下面矩阵的 LU 分解

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 0\end{bmatrix}$$

分别用两种方法调用 MATLAB 下的 lu() 函数，则可以得出的不同结果。

```
>>  A=[1, 2, 3; 4, 5, 6; 7, 8, 0];
>>  [L1,U1]=lu(A)
L1 =   0.1429    1.0000         0
       0.5714    0.5000    1.0000
       1.0000         0         0
U1 =   7.0000    8.0000         0
            0    0.8571    3.0000
            0         0    4.5000
>>  L1*U1
ans =   1     2     3
        4     5     6
        7     8     0

>>  [L, U, P] = lu(A)
L =   1.0000         0         0
      0.1429    1.0000         0
      0.5714    0.5000    1.0000
U =   7.0000    8.0000         0
           0    0.8571    3.0000
```




```
            0         0    4.5000
P =    0      0      1
       1      0      0
       0      1      0
>>   L*U   % Not A
ans  = 7      8      0
       1      2      3
       4      5      6
>>  inv(P)*L*U   % Equals to A
ans  = 1      2      3
       4      5      6
       7      8      0
```

注意，这里得出的 P 矩阵不是一个单位阵，所以在进行计算时由于考虑主元素的原因对原来的排列进行了改动，这样前一种方法得出的 L 也不是一个真正的下三角矩阵。在后一种调用方式中，注意 $LU \neq A$ 。

如果 A 为对称矩阵，则可以用类似的方法对之进行分解，这样可以将原来矩阵 A 分解成

$$A = DD^T = \begin{bmatrix} d_{11} & & & \\ d_{21} & d_{22} & & \\ \vdots & \vdots & \ddots & \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{bmatrix} \begin{bmatrix} d_{11} & d_{21} & \cdots & d_{n1} \\ & d_{22} & \cdots & d_{n2} \\ & & \ddots & \vdots \\ & & & d_{nn} \end{bmatrix} \tag{3.3.9}$$

其中 D 矩阵可以形象地理解为原 A 矩阵的平方根。对该对称矩阵进行分解可以采用 Cholesky 分解算法，其具体叙述如下

$$d_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} d_{ik}^2}, \quad d_{ij} = \frac{a_{ij} - \sum\limits_{k=1}^{j-1} d_{ik} d_{jk}}{l_{jj}}, \ (j < i) \tag{3.3.10}$$

MATLAB 提供了 chol() 函数来求取矩阵的 Cholesky 分解矩阵 D，该函数的调用格式可以写成

```
[D, P] = chol(A)
```

式中返回的 D 为 Cholesky 分解矩阵，且 $A = DD^T$，而 P-1 为 A 矩阵中正定的子矩阵的阶次，如果 A 为正定矩阵，则返回 $P = 0$。当然也可以直接由 D=chol(A) 来调用该函数，这时要求 A 为一个正定矩阵。如果 A 不是正定矩阵，则这样调用将给出一个错误信息。

例 3.7　重新考虑例 3.2 中给出的矩阵 A，可见该矩阵为一个对称矩阵，调用 MATLAB 的 chol() 函数可以得出如下的结果

```
>>  [D, P]=chol(A)
```




```
D = 3.1623    1.5811    1.2649    0.9487    0.6325    0.3162
         0    2.7386    1.0954    0.9129    0.7303    0.5477
         0         0    2.6833    1.0435    0.8944    0.7454
         0         0         0    2.6791    1.0451    0.9041
         0         0         0         0    2.6785    1.0410
         0         0         0         0         0    2.6727
P =      0
```

可见式中 $P=0$，这将表示 A 阵为一个正定矩阵。如果试图对一个负定矩阵，例如 $-A$ 作 Cholesky 分解，则将得出如下的信息

```
>>  [D,P]=chol(-A)
D =      []
P =      1

>>  D=chol(-A)
??? Error using ==> chol
Matrix must be positive definite.
```

其中前一种调用下将返回一个空矩阵 D，但不会中止程序的运行，而在后一种调用格式下，将给出错误信息，同时给出一声蜂鸣警告，并中止程序的运行。

3.3.3 矩阵的奇异值分解

矩阵的奇异值也可以看成是矩阵的一种测度，对任意的 $n\times m$ 矩阵 A 来说，总有

$$A^TA\geq 0,\quad AA^T\geq 0 \tag{3.3.11}$$

且有

$$\mathrm{rank}\{A^TA\}=\mathrm{rank}\{AA^T\}=\mathrm{rank}\{A\} \tag{3.3.12}$$

进一步可以证明，A^TA 与 AA^T 有相同的非零特征值 λ_i，且相同的非零特征值总是为正数。在数学上把这些非零的特征值的平方根称作矩阵 A 的奇异值，记 $\sigma_i\{A\}=\sqrt{\lambda_i\{A^TA\}}$。

矩阵的奇异值大小通常决定矩阵的性态，如果矩阵的奇异值变化特别大，则矩阵中某个参数有一个微小的变化将严重影响到原矩阵的参数，如其特征值的大小，这样的矩阵又称为病态矩阵，有时也称为奇异矩阵。

例 3.8 考虑下面的著名矩阵 A 来演示矩阵奇异值的作用 [6]

$$A=\begin{bmatrix}1 & 1\\ \mu & 0\\ 0 & \mu\end{bmatrix}$$

由于矩阵的奇异值是通过矩阵 A^TA 的特征值来求出的，而对于前面给出的矩阵 A 来说，有

$$A^TA=\begin{bmatrix}1+\mu^2 & 1\\ 1 & 1+\mu^2\end{bmatrix}$$

如果 μ 的值特别小，将有 $\mu^2\ll\mu$，以致于在机器允许的精度下 $\mu^2+1\to 1$，则 A^TA 可以近似地看出一个矩阵元素全部为 1 的 2×2 矩阵，这样就可以得出 A^TA 矩阵特征值为 2 和 0，亦即原矩阵 A 的奇异值为 $\sqrt{2}$ 和 0 这样的结论。

如果 μ 在计算中不可以忽略，这样显然可以看出原矩阵的秩 $\text{rank}\{A\}=2$，而使用数值方法后得出错误的结论 $\text{rank}\{A\}=1$。出现这样错误的原因是在计算中不必要地引入了 μ^2，从而使得其结果被错误地舍入了，所以只有通过奇异值分解的方法才能正确地求出矩阵 A 的秩。

假设 A 矩阵为 $n\times m$ 矩阵，且 $\text{rank}\{A\}=r$, 则 A 矩阵可以分解为

$$A=L\begin{bmatrix}\Delta & 0\\ 0 & 0\end{bmatrix}M \tag{3.3.13}$$

其中 L 和 M 为正交矩阵，$\Delta=\text{diag}\{\sigma_1,\cdots,\sigma_r\}$ 为对角矩阵，且其对角元素均不为 0。

MATLAB 提供了直接求取矩阵奇异值分解的函数，其调用方式为

```
[U, A1, V] = svd(A);
```

其中 A 为原始矩阵，返回的 A1 为对角矩阵，而 U 和 V 均为变换矩阵，并满足 $A=U\,A_1\,V^T$。

用上面的极端例子来理解奇异值的概念，假设式中的 μ 取了一个极小的数值 $\mu=5\times\text{eps}$, 这时可以构造出 A 矩阵，并求出其秩和奇异值来

```
>>  A=[1,1; 5*eps,0; 0,5*eps];
>>  sqrt(eig(A'*A))
ans =       0
       1.4142
>>  [u,g,v]=svd(A)
u = 1.0000    0.0000    0.0000
    0.0000   -0.7071    0.7071
    0.0000    0.7071    0.7071
g = 1.4142         0
         0    0.0000
         0         0
v = 0.7071   -0.7071
    0.7071    0.7071
>>  g(2,2)
ans = 1.1102e-015
>>  rank(A)
ans = 2
```

从这一例子可以看出，由求取特征值的方法得出的奇异值是不正确的，而由奇异值分解的算法将得出正确的奇异值，因为在前一种方法中引入了 μ^2 的值而最终被舍入，故结果是不正确的。在奇异值分解中可见 A 阵的最小奇异值并不是 0，而是 1.1102×10^{-15} 这样的小数，故这时此矩阵的秩为 2，而不是 1，可见采用奇异值分解的工具可以正确地得出矩阵的秩。

矩阵最大奇异值 $\sigma_{\max}$ 和最小奇异值 $\sigma_{\min}$ 的比值又称为该矩阵的条件数，记作 $\text{cond}\{A\}$, 即 $\text{cond}\{A\}=\sigma_{\max}/\sigma_{\min}$, 矩阵的条件数越大，则对参数变化越敏感。矩阵的最大和最小奇异值还分别经常记作 $\bar{\sigma}\{A\}$ 和 $\underline{\sigma}\{A\}$。在 MATLAB 下也提供了求取矩阵条件数的函数 cond(), 其调用格式为



```
cond(A)
```

例 3.9 考虑例 3.2 中给出的 A 矩阵，如果调用 MATLAB 中给出的矩阵奇异值分解函数 svd()，则可以容易地求出 U, Γ 和 V 矩阵，并可以容易地求出该矩阵的条件数。

```
>>  [U, G, V]=svd(A)
U =   0.4566   -0.3534    0.1227    0.5642    0.5742   -0.0600
      0.2187   -0.5343    0.5642   -0.1227   -0.5273   -0.2352
     -0.0778   -0.5721   -0.1227   -0.5642    0.3391    0.4673
     -0.3534   -0.4566   -0.5642    0.1227   -0.0600   -0.5742
     -0.5343   -0.2187    0.1227    0.5642   -0.2352    0.5273
     -0.5721    0.0778    0.5642   -0.1227    0.4673   -0.3391
G =  18.2911         0         0         0         0         0
           0   18.2911         0         0         0         0
           0         0    8.5440         0         0         0
           0         0         0    8.5440         0         0
           0         0         0         0    7.5786         0
           0         0         0         0         0    7.5786
V =   0.5774    0.0000   -0.0832    0.5713    0.5773    0.0013
      0.5000   -0.2887    0.5713    0.0832   -0.4994   -0.2898
      0.2887   -0.5000    0.0832   -0.5713    0.2876    0.5006
      0.0000   -0.5774   -0.5713   -0.0832    0.0013   -0.5773
     -0.2887   -0.5000   -0.0832    0.5713   -0.2898    0.4994
     -0.5000   -0.2887    0.5713    0.0832    0.5006   -0.2876
>>  cond(A)
ans =     2.4135
>>  G(1,1)/G(6,6)
ans =     2.4135
>>  B = A' *A;  sqrt(eig(B))
ans =    7.5786
         7.5786
         8.5440
         8.5440
        18.2911
        18.2911
>>  ans(6)/ans(1)
ans =     2.4135
```

这里用到了三种方式来求取矩阵的条件数，其中第一种方法是直接调用 cond() 函数来求的，而第二种方式是根据条件数的定义直接由矩阵奇异值的最大值与最小值的比值来求的，而第三种方法首先求出 A^TA 矩阵的特征值，再由它开平方得出原矩阵的奇异值，最后由矩阵的奇异值求出该矩阵的条件数，这三种方式得出的结果是一致的。

例 3.10 对于 $n \neq m$ 的矩阵 A 来说，也可以对之作奇异值分解，例如

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix}$$

如果使用如下命令则可以得出

```
>> A = [1, 3, 5, 7; 2, 4, 6, 8];
>> [U, G, V] = svd(A)
U =     0.6414    0.7672
        0.7672   -0.6414
G =    14.2691         0         0         0
             0    0.6268         0         0
V =     0.1525   -0.8226    0.5477         0
        0.3499   -0.4214   -0.7303    0.4082
        0.5474   -0.0201   -0.1826   -0.8165
        0.7448    0.3812    0.3651    0.4082
>> U*G*V'
ans =   1.0000    3.0000    5.0000    7.0000
        2.0000    4.0000    6.0000    8.0000
>> norm(A-ans)
ans =  3.2256e-015
```

对这个例子进行逆运算，即 $U\Gamma V^T$ ，则可以还原成原来的 A 矩阵，由前面的分析可见，这样得出的矩阵误差是很小的。

3.4 矩阵的特征值与特征向量

对一个矩阵 A 来说，如果存在一个非零的向量 x ，且有一个标量 λ 满足

$$Ax = \lambda x \tag{3.4.1}$$

则称 λ 为 A 矩阵的一个特征值，而 x 为对应于特征值 λ 的特征向量，严格说来， x 应该称为 A 的右特征向量。如果矩阵 A 的特征值不包含重复的值，则对应的各个特征向量为线性独立的，这样由各个特征向量可以构成一个非奇异的矩阵，如果用它对原始矩阵作相似变换，则可以得出一个对角矩阵。矩阵的特征值与特征向量由 MATLAB 提供的函数 eig() 可以容易地求出，该函数的调用格式为

```
[V, D] = eig(A)
```

其中 A 为要处理的矩阵，D 为一个对角矩阵，其对角线上的元素为矩阵 A 的特征值，而每个特征值对应的 V 矩阵的列为该特征值的特征向量，该矩阵是一个满秩矩阵。MATLAB 的矩阵特征值的结果满足 $AV = VD$ ，且每个特征向量各元素的平方和 (即 2 范数) 均为 1 。如果调用该函数时只给出一个返回变量，则将只返回矩阵 A 的特征值。即使 A 为复数矩阵，也照样可以由 eig() 函数得出其特征值与特征向量矩阵的。

矩阵的特征值等概念在控制系统的研究中还是很重要的，例如要判断一个线性系统的稳定性的一种最有效的方法是直接求出系统所有的极点，然后根据极点的分布情况来确定系统的稳定性。系统的极点求取当然可以由矩阵特征值的求解来完成。

矩阵特征值的求解算法是多种多样的，最常用的有求解实对称矩阵特征值与特征向量的 Jacobi 算法，有原点平移 QR 分解法与两步 QR 算法，矩阵的特征值与特征向量的求解有许多标准的子程序或程序库可以直接调用，如著名的 EISPACK 软件包 [3, 11] 等。



例 3.11　考虑例 3.2 中给出的矩阵 A，如果调用 eig() 函数，则可以获得矩阵 A 的特征值与特征向量

```
>> [v, d] = eig(A)
v =      Columns 1 through 4
      -0.3853 + 0.1350i  -0.3853 - 0.1350i  -0.4077 + 0.0202i  -0.4077 - 0.0202i
      -0.4012 - 0.0757i  -0.4012 + 0.0757i  -0.0202 - 0.4077i  -0.0202 + 0.4077i
      -0.3096 - 0.2661i  -0.3096 + 0.2661i   0.4077 - 0.0202i   0.4077 + 0.0202i
      -0.1350 - 0.3853i  -0.1350 + 0.3853i   0.0202 + 0.4077i   0.0202 - 0.4077i
       0.0757 - 0.4012i   0.0757 + 0.4012i  -0.4077 + 0.0202i  -0.4077 - 0.0202i
       0.2661 - 0.3096i   0.2661 + 0.3096i  -0.0202 - 0.4077i  -0.0202 + 0.4077i
     Columns 5 through 6
       0.3418 + 0.2233i   0.3418 - 0.2233i
      -0.4076 - 0.0224i  -0.4076 + 0.0224i
       0.3642 - 0.1844i   0.3642 + 0.1844i
      -0.2233 + 0.3418i  -0.2233 - 0.3418i
       0.0224 - 0.4076i   0.0224 + 0.4076i
       0.1844 + 0.3642i   0.1844 - 0.3642i
 d =   Columns 1 through 4
      14.4641 +11.1962i        0                  0                  0
            0            14.4641 -11.1962i        0                  0
            0                  0            8.0000 + 3.0000i         0
            0                  0                  0            8.0000 - 3.0000i
            0                  0                  0                  0
            0                  0                  0                  0
     Columns 5 through 6
            0                  0
            0                  0
            0                  0
            0                  0
      7.5359 + 0.8038i         0
            0            7.5359 - 0.8038i
  >>  max(norm(A*v-v*d))
  ans =  1.5594e-014

  >>  eig(A)
  ans =  14.4641 +11.1962i
         14.4641 -11.1962i
          8.0000 + 3.0000i
          8.0000 - 3.0000i
          7.5359 + 0.8038i
          7.5359 - 0.8038i
```

可见在前面的例子中两次调用了 eig() 函数，但由于返回参数个数不一致，所以前面的调用返回矩阵 A 的特征值与特征向量，而后面的调用只返回了矩阵 A 的特征值而不返回特征向量。另外，返回特征值的格式因返回变量个数不同而不同。



若想由 C 或 FORTRAN 语句从最底层编程，则需要编写相当大的程序才可以完成此计算。当然求取矩阵特征值与特征向量的标准子程序可以由文献 [11] 中得出。

例 3.12　考虑一个矩阵

$$A = \begin{bmatrix} 1. & 100 & 10000 \\ 0.01 & 1 & 100 \\ 0.0001 & 0.1 & 1 \end{bmatrix}$$

分析 A 矩阵的性质

```
>>  A = [1, 100, 10000; 0.01, 1, 100; 0.0001, 0.1, 1];
>>  [V1, D1] = eig(A);  V1
V1 =     1.0000    -0.9999    -0.9999
         0.0000    -0.0100    -0.0100
        -0.0001    -0.0003     0.0004
>>  cond(V1)
ans =  3.6517e+003
```

可见特征向量矩阵 V_1 的条件数相当大，可以被认为是病态矩阵，所以再利用得出的 V_1 进行操作是很不合适的。在实际计算中，往往要对原始矩阵作一种特殊的相似变换，这种变换称为均衡变换，它可以通过 balance() 函数的调用来完成，即引入一个矩阵 T，使得变换后的矩阵 B 满足 $TBT^{-1} = A$。

```
>>  [T, B]=balance(A)
T =  1.0e+003 *
    4.0960         0         0
         0    0.0160         0
         0         0    0.0005
B =  1.0000    0.3906    1.2207
     2.5600    1.0000    3.1250
     0.8192    3.2000    1.0000
>>  [V2, D2] = eig(B); V2
V2 = 0.7736   -0.2771   -0.2389
     0.0000   -0.7095   -0.6116
    -0.6337   -0.6480    0.7543
>>  cond(V2)
ans =  2.3503
```

可以看出，经过均衡变换之后的矩阵 B 的可读性更强，因为矩阵元素之间的差异不像原来 A 矩阵的那样悬殊了。对经过均衡变换的矩阵求取特征值与特征向量，则得出的特征向量 V_2 的结果也是比较均衡的，其条件数也大大地减小了。值得注意的是，这两种方法得出的矩阵特征值是一致的，但是 V_2 是 B 矩阵的特征向量矩阵，而不是 A 矩阵的特征向量矩阵，在使用时不要对之产生混淆。在实际运算中，如果原矩阵的特征向量条件数过大，则采用均衡变换还是必要的，但使用时还是应该慎重。考虑下面一个例子

$$A = \begin{bmatrix} 3 & -2 & -0.9 & 2*\epsilon \\ -2 & 4 & -1 & -\epsilon \\ -\epsilon/4 & \epsilon/2 & -1 & 0 \\ -0.5 & -0.5 & 0.1 & 1 \end{bmatrix}$$

其中 ϵ 表示机器能表示的精度，前面介绍过在 MATLAB 下 ϵ =eps $= 2.2204 \times 10^{-16}$。如果不对矩阵进行任何处理而直接求取特征值及特征向量，将得出下面的结果



```
>> A=[3, -2, -0.9, 2*eps; -2, 4, -1, -eps;
     -eps/4, eps/2, -1, 0; -0.5, -0.5, 0.1, 1];
>> [V1, D1]=eig(A);  norm(A*V1-V1*D1)
ans =    1.2439
>> [V2, D2]=eig(A,'nobalance');  norm(A*V2-V2*D2)
ans =  2.9957e-015
>> norm(D1-D2)
ans =  1.7764e-015
```

在前面的计算中，首先自动采用均衡的方法得出矩阵的特征值 D_1 和特征向量矩阵 V_1，然后求 $AV_1 - V_1D_1$ 的范数时发现，该矩阵并不是和定义中给定的那样为一个零矩阵，这样就可以知道，得出的矩阵特征向量矩阵有很大的误差，如果关闭 eig() 函数中的均衡变换选项，则得出的 V_2 和 D_2 满足 $AV_2 - V_2D_2 = 0$，所以对此例来说，如果进行均衡变换将得出错误的特征向量矩阵。上面的命令末尾求出了两种方法所得出的特征值的差异，可以看出，是否经过均衡变换对特征值的求取及精度几乎没有任何影响。

MATLAB 还提供了求取广义特征值的方法，所谓广义特征值的概念如下，假设存在一个标量 λ 和一个非零向量 x，使得

$$Ax = \lambda Bx \tag{3.4.2}$$

成立，则 λ 称为广义特征值，而 x 向量称为广义特征向量。事实上，普通的矩阵特征值问题可以看成是广义特征值问题的一个特例，因为若假定 $B = I$ 为单位阵，则式 (3.4.2) 中的形式可以转化成普通矩阵特征值问题。

如果 B 矩阵为一个非奇异矩阵，则上面的方程可以容易地转换成一般的特征值问题

$$B^{-1}Ax = \lambda x \tag{3.4.3}$$

即 λ 和 x 分别为 $B^{-1}A$ 矩阵的特征值和特征向量。但一般情况下不能随便假设 B 阵为非奇异的，所以文献 [7] 中给出了广义特征值问题的 QZ 算法。在 MATLAB 中给出的 eig() 函数可以直接用来求取矩阵的广义特征值和特征向量，这时的调用格式为

```
[V, D] = eig(A, B)
```

这一函数仍返回一个满秩的特征向量矩阵 V 及一个对角型特征值矩阵 D，满足 $AV = BVD$。

例 3.13　假设矩阵 A 和 B 如下给出

$$A = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 6 & -1 & -2 \\ 5 & -1 & 2 & 3 \\ -3 & -4 & 1 & 10 \\ 5 & -2 & -3 & 8 \end{bmatrix}$$

则使用下列命令可以求出矩阵的广义特征值和特征向量

```
>> A=[5, 7, 6, 5; 7, 10, 8, 7;  6, 8, 10, 9; 5, 7, 9, 10];
>> B=[2, 6, -1, -2; 5, -1, 2, 3; -3, -4, 1, 10; 5, -2, -3, 8];
```




```
>>  [V, D] = eig(A, B)
V =   0.2224 - 0.0000i   0.5157 - 0.0188i  -0.0188 + 0.5157i   0.8328 + 0.0000i
      0.5985 - 0.0000i  -0.1244 + 0.1375i   0.1375 - 0.1244i  -0.5072 + 0.0000i
      0.4800 - 0.0000i  -0.3400 - 0.5588i  -0.5588 - 0.3400i  -0.1929 - 0.0000i
      0.6016 - 0.0000i   0.0603 + 0.5175i   0.5175 + 0.0603i   0.1098 + 0.0000i
D =   4.7564 + 0.0000i          0                  0                  0
            0           0.0471 + 0.1750i          0                  0
            0                  0           0.0471 - 0.1750i          0
            0                  0                  0              -0.0037
>> norm(A*V-B*V*D)
ans =   9.6583e-015
```

3.5 矩阵求逆与线性方程求解

3.5.1 矩阵求逆运算与线性方程求解

矩阵求逆运算往往和线性代数方程的求解有关，考虑下面给出的线性代数方程

$$Ax = B \tag{3.5.1}$$

式中 A 和 B 为相容维数的矩阵，而 x 亦为矩阵，它称为方程的解。

事实上，如果给定的 B 矩阵为单位阵，且 A 为非奇异的方阵，则可以立即得出

$$x = A^{-1}B \tag{3.5.2}$$

并称 A^{-1} 为 A 矩阵的逆矩阵。对一个已知的 $n \times n$ 非奇异方阵 A 来说，如果有一个同样大小的 C 矩阵满足

$$AC = CA = I \tag{3.5.3}$$

式中 I 为单位阵，则称 C 矩阵为 A 矩阵的逆矩阵，并记作 $C = A^{-1}$ 。这时式 (3.5.1) 中方程的解可以写成 $x = A^{-1}B$ 。线性方程的求解和矩阵求逆的算法是多种多样的，比较常用的有全 (列) 主元素 Gauss 消去法、LU 分解法、基于奇异值分解的方法等，MATLAB 提供了一个求取逆矩阵的函数 inv()，其调用格式为

```
B=inv(A)
```

通过这一函数的调用就可以直接由给出的 A 矩阵求出其逆矩阵 B 来。在 MATLAB 下式 (3.5.1) 中的解可以由 x=inv(A)*B 求出，也可以简单地由 x=A/B 求出。但是 inv() 函数的调用也有值得注意之处，例如如果 A 矩阵为奇异的或接近奇异，则利用此函数有可能产生错误的结果。

例 3.14 考虑例 3.2 中给出的矩阵，如果调用 MATLAB 的矩阵求逆函数 inv()，则可以立即得出该矩阵的逆矩阵来

```
>>  inv(A)
```




```
ans =    0.0947   -0.0470   -0.0131   -0.0021    0.0018    0.0038
        -0.0038    0.0947   -0.0470   -0.0131   -0.0021    0.0018
        -0.0018   -0.0038    0.0947   -0.0470   -0.0131   -0.0021
         0.0021   -0.0018   -0.0038    0.0947   -0.0470   -0.0131
         0.0131    0.0021   -0.0018   -0.0038    0.0947   -0.0470
         0.0470    0.0131    0.0021   -0.0018   -0.0038    0.0947
```

例 3.15 如果原始矩阵 A 为下面的奇异矩阵

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$$

则调用 MATLAB 的矩阵求逆函数可以得出下面的结果

```
>>  A=[1, 2, 3; 4, 5, 6; 7, 8, 9];
>>  B = inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.937385e-018
B =   1.0e+016 *
      0.3152   -0.6304    0.3152
     -0.6304    1.2609   -0.6304
      0.3152   -0.6304    0.3152
```

可见这时给出一个警告信息，提示用户该矩阵接近于奇异(事实上 a 矩阵是一个奇异矩阵，但通过数值算法后得出的是接近奇异的结论)，并提示得出的逆矩阵可能是不正确的，最后还列出了逆矩阵的结果。当然从结果看来显然是没有意义的。

3.5.2 矩阵的广义逆

如果用户确实需要得出原来奇异矩阵的一种“逆 ”阵，就需要使用广义逆的概念了。对要研究的矩阵 A，如果存在一个矩阵 N，它满足

$$ANA = A \tag{3.5.4}$$

则 N 矩阵称为 A 的广义逆矩阵，记作 $N = A^{-}$。MATLAB 提供了求取矩阵广义逆的函数 pinv()，该函数的调用格式为

```
B = pinv(A, tol)
```

其中 tol 为判 0 用误差限，如果省略此参数，则判 0 用误差限选用机器的精度 eps，这时将返回 A 的广义逆矩阵 B 。

可以重新考虑前面例子中的奇异矩阵，如果调用了 pinv() 函数，并对结果进行检验，则会得出以下的结果

```
>>  iA = pinv(A)
ans = -0.6389   -0.1667    0.3056
```

```
    -0.0556     0.0000     0.0556
     0.5278     0.1667    -0.1944
>> iA*A
ans =  0.8333     0.3333    -0.1667
       0.3333     0.3333     0.3333
      -0.1667     0.3333     0.8333
>> A*iA
ans =  0.8333     0.3333    -0.1667
       0.3333     0.3333     0.3333
      -0.1667     0.3333     0.8333
>> A*iA*A
ans =  1.0000     2.0000     3.0000
       4.0000     5.0000     6.0000
       7.0000     8.0000     9.0000
>> iA*A*iA
ans = -0.6389    -0.1667     0.3056
      -0.0556     0.0000     0.0556
       0.5278     0.1667    -0.1944
```

可以看出，这样得出的逆矩阵元素不再是没有意义的数据了，但对结果作 AB 和 BA 运算，则得出的两个矩阵不再是单位阵了。尽管得出的"逆"矩阵不满足常规逆矩阵的一般要求，但它满足 $AA^{-}A=A$ 及 $A^{-}AA^{-}=A^{-}$ 两个条件。

前面考虑的都是 A 为方阵的情况，如果 A 阵是 $n\times m$ 一个长方形矩阵 (其中 $m\neq n$)，则用 pinv() 函数也可以求出 A 阵的广义逆矩阵，如果定义下面的范数最小化指标

$$\min_{x}||Ax-B|| \tag{3.5.5}$$

则满足这一指标的解共有无穷多个， MATLAB 的 pinv() 函数是众多这样的解中的一个，但这一个解有其特殊意义，因为它是唯一的。可以证明，对一个给定的矩阵 A, 存在一个唯一的矩阵 M 使得下面三个条件同时成立

1) $AMA=A$
2) $MAM=M$
3) AM 与 MA 均为对称矩阵

这样矩阵 M 称为矩阵 A 的 Moore-Penrose 广义逆矩阵，记作 $M=A^{+}$。从上面的三个条件中可以看出，第一个条件和一般广义逆的定义也是一样的，所不同的是它还要求满足第二和第三个条件，这样就会得出唯一的广义逆矩阵了。更进一步地，如果对复数矩阵 A 来说，若得出的广义逆矩阵的第三个条件扩展为 $M^{*}A$ 与 AM^{*} 均为 Hermit 矩阵，则这样构造的矩阵也是唯一的。MATLAB 的 pinv() 函数得出的就是这样的广义逆矩阵。

除了上面的求解方法以外，另外还可以把方程的解简单地写成 y=A\ B 。这样得出的解同样能够极小化式 (3.5.5) 中所示的性能指标，它和由广义逆求解的方法的区别主要在于，用这种方法获得的解含有多个零值。

例 3.16 考虑下面的长方形方程

$$\begin{bmatrix} 6 & 1 & 4 & 2 & 1 \\ 3 & 0 & 1 & 4 & 2 \\ -3 & -2 & -5 & 8 & 4 \end{bmatrix} x = \begin{bmatrix} 11 \\ 6 \\ 4 \end{bmatrix}$$

可以使用下列 MATLAB 函数和命令来对该方程作相应的分析

```
>> A = [6, 1, 4, 2, 1; 3, 0, 1, 4, 2; -3, -2, -5, 8, 4];
>> B = [11; 6; 4];
>> rank(A)
ans =     2
>> x1 = pinv(A)*B
x1 =  0.9625
      0.0682
      0.4573
      0.8739
      0.4370
>> norm(A*x1-b)
ans =    2.1381
>> x2=A\B
Warning: Rank deficient, rank = 2  tol =   1.0175e-014
x2 =  1.3333
           0
           0
      0.9286
           0
>> norm(A*x2-b)
ans =   2.1381
```

可见，A 矩阵不是满秩矩阵，如果调用 MATLAB 下的 pinv() 函数则可以得出一组方程的解 x1. 将 x1 代入则可以求出范数值，若直接采用右除的方法也可以得出一组解 x2, 其范数与 x1 的一致，但是在解中含有很多零值. 下面对得出的广义逆矩阵的性质进行分析

```
>> iA = pinv(A)
iA =  0.0730    0.0413   -0.0221
      0.0108    0.0020   -0.0156
      0.0459    0.0178   -0.0385
      0.0327    0.0431    0.0638
      0.0164    0.0215    0.0319
>> iA*A*iA
ans =  0.0730    0.0413   -0.0221
       0.0108    0.0020   -0.0156
       0.0459    0.0178   -0.0385
       0.0327    0.0431    0.0638
       0.0164    0.0215    0.0319
>> norm(iA-ans)   % Testing for iA*A*iA=iA
ans =  3.0278e-017
```

```
>> norm(A*iA*A-A) %Testing for A*iA*A=A
ans =  2.8883e-015
>> norm(iA*A-A'*iA')  % Testing for iA*A symmetric
ans =  1.4543e-016
>> norm(A*iA-iA'*A')  % Testing for A*iA symmetric
ans =  6.7987e-017
```

可见由 MATLAB 直接得出的广义逆矩阵为 Moore-Penrose 逆。下面考虑对 A^+ 再求一次广义逆，并观察其结果

```
>> iiA = pinv(iA)
iiA =  6.0000    1.0000    4.0000    2.0000    1.0000
       3.0000    0.0000    1.0000    4.0000    2.0000
      -3.0000   -2.0000   -5.0000    8.0000    4.0000
>>  norm(iiA-A)
ans =  5.9998e-015
```

由前面给出的结果可见对一个矩阵的广义逆再求一次广义逆，则将还原成原来的矩阵，亦即 $(A^+)^+=A$ 。

3.5.3 Kronecker 积与方程求解

对于一类特殊线性代数的方程

$$AX=C \tag{3.5.6}$$

其中 A 为 $n\times n$ 矩阵，且 C 为 $n\times m$ 矩阵，为叙述方便起见可以将上面各个矩阵的参数记成

$$X=\begin{bmatrix} x_1 & x_2 & \cdots & x_m \\ x_{m+1} & x_{m+2} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{(n-1)m+1} & x_{(n-1)m+2} & \cdots & x_{nm} \end{bmatrix},\quad C=\begin{bmatrix} c_1 & c_2 & \cdots & c_m \\ c_{m+1} & c_{m+2} & \cdots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(n-1)m+1} & c_{(n-1)m+2} & \cdots & c_{nm} \end{bmatrix} \tag{3.5.7}$$

当然该方程的解可以由 $X=A^{-1}C$ 求出，同时由 MATLAB 函数直接可以求出该方程的解 `X = inv(A)*C` 。往往我们更希望将上面的方程转换成方程右边为一个列向量，且方程的解也由一个列向量来表示，则需要进行特殊的变换。

可以证明，该方程可以变换成

$$(A\otimes I_m)x=c \tag{3.5.8}$$

式中 $\otimes$ 表示两个矩阵的 Kronecker 乘积，而 x 和 c 分别为向量，其表示方法为

$$x^T=[x_1\ \ x_2\ \ \cdots\ \ x_{nm}],\quad c^T=[c_1\ \ c_2\ \ \cdots\ \ c_{nm}] \tag{3.5.9}$$

这样原方程的解 x 就可以容易地求出了。



例 3.17　假设有两个矩阵

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix},\quad C=\begin{bmatrix}1&5\\2&4\\4&6\end{bmatrix}$$

这样的线性代数方程可以由下面两种方法解出

```
>>  A=[1 2 3;4 5 6; 7 8 0];
>>  C=[1 5; 2 4; 4 6];
>>  x1 = inv(A)*C  % Direct Method
x1 = -0.4444   -6.0000
      0.8889    6.0000
     -0.1111   -0.3333
>>  norm(A*x1-C)
ans =  6.5393e-015
>>  A1=kron(A,eye(2))  % Indirect Method
A1 =  1     0     2     0     3     0
      0     1     0     2     0     3
      4     0     5     0     6     0
      0     4     0     5     0     6
      7     0     8     0     0     0
      0     7     0     8     0     0
>>  c1=[C(1,:) C(2,:) C(3,:)]'
c1 =  1
      5
      2
      4
      4
      6
>>  x2 = inv(A1)*c1
x2 = -0.4444
     -6.0000
      0.8889
      6.0000
     -0.1111
     -0.3333
>>  x3 = [x2(1:2)'; x2(3:4)'; x2(5:6)']
x3 = -0.4444   -6.0000
      0.8889    6.0000
     -0.1111   -0.3333
>>  norm(A*x3-C)
ans = 6.5393e-015
```

可见，用这样两种方法得出的解是完全相同的，虽然两种方法所用到的解的结构有所不同，但通过简单的变换就可以得出满足原始方程的解 x3 。



当然上面演示的算法并不是利用 Kronecker 乘积的主要目的，理解了上述的变换之后，还可以用 Kronecker 积来处理更复杂的方程，比如下面给出的 Lyapunov 方程

$$AX + XB = C \tag{3.5.10}$$

式中 A 为 $n\times n$ 矩阵， B 为 $m\times m$ 矩阵，利用 Kronecker 乘积的表示方法，上面的方程可以写成

$$(A\otimes I_m + I_n\otimes B^T)x = c \tag{3.5.11}$$

例 3.18 假设式 (3.5.10) 中的 A, B 和 C 矩阵分别为

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix},\quad B=A^T,\quad C=\begin{bmatrix}1&5&4\\5&6&7\\4&7&9\end{bmatrix}$$

则可以由下面的 MATLAB 语句求出该方程的解

```
>> A=[1 2 3;4 5 6; 7 8 0];  B=A';
>> C=[1, 5, 4; 5, 6, 7; 4, 7, 9];
>> A0=kron(A,eye(3))+kron(eye(3),B')
A0 =  2    2    3    2    0    0    3    0    0
      4    6    6    0    2    0    0    3    0
      7    8    1    0    0    2    0    0    3
      4    0    0    6    2    3    6    0    0
      0    4    0    4   10    6    0    6    0
      0    0    4    7    8    5    0    0    6
      7    0    0    8    0    0    1    2    3
      0    7    0    0    8    0    4    5    6
      0    0    7    0    0    8    7    8    0
>> c=[C(1,:) C(2,:) C(3,:)];
>> x0 = inv(A0)*c'
x0 = 1.5556
    -1.1111
     0.3889
    -1.1111
     1.2222
     0.2222
     0.3889
     0.2222
     0.3889
>> D=[x0(1:3)'; x0(4:6)'; x0(7:9)']
D =  1.5556   -1.1111    0.3889
    -1.1111    1.2222    0.2222
     0.3889    0.2222    0.3889
>> C0=A*D+D*A'
C0 = 1.0000    5.0000    4.0000
     5.0000    6.0000    7.0000
```

```
    4.0000    7.0000    9.0000
>> norm(C0-C)
ans =  8.2709e-015
```

上面求出了需要的 Lyapunov 方程的解，并将该解代入了原始的方程，可见得出的结果满足原始方程，其误差也是非常小的，故该算法可以以相当高的精度求解相应的方程。事实上，MATLAB 中也提供了 Lyapunov 方程的求解函数 lyap()，在第 5 章中还将详细介绍该函数，这里直接使用该函数得出方程的解

```
>> X=lyap(A,A',-C)
X =  1.5556   -1.1111    0.3889
    -1.1111    1.2222    0.2222
     0.3889    0.2222    0.3889
>> norm(A*X+X*A'-C)
ans = 1.0400e-014
```

可见直接调用 lyap() 函数得出的方程解和前面的方法得出的是几乎完全一致的，两者之间有差异是因为所采用的算法不同，对此例来说似乎由 Kronecker 积的算法得出解的精度稍高一些。

3.6 稀疏矩阵的处理

在实际的矩阵运算中，往往要用到一些特殊的矩阵，这些矩阵的很多元素的值都是 0，只有少部分的元素为非 0 的，例如一个单位矩阵，只有对角线上的元素为非 0 元素，而其它所有元素的值全是 0，这样再用常规矩阵的形式来表示该矩阵是很费空间的。于是很自然地出现了稀疏矩阵 (sparse matrix) 的表示方法。

MATLAB 4.0 中成功地引入了稀疏矩阵的表示方法[4]，并定义了大量的稀疏矩阵处理函数。假设用户想产生一个稀疏形式表示的单位矩阵，则可以调用下面的命令

```
A=speye(100);
```

这样就会产生一个 100×100 的单位矩阵，如果想显示 A 矩阵的内容，则可以在 MATLAB 提示符下键入 A，这样将会把 A 矩阵的内容显示出来

```
>> A
A =     (1,1)        1
        (2,2)        1
        (3,3)        1
        (4,4)        1
        (5,5)        1
        (6,6)        1
         ...   ...  ...
      (100,100)      1
```

如果按照常规的方法生成一个单位矩阵 B，则再用 who 命令来显示工作空间中的变量，将得出如下的结果

```
>> B = eye(100); whos
     Name      Size       Elements     Bytes    Density   Complex
```

```
 A      100 by 100          100      1600    0.0100      No
 B      100 by 100        10000     80000      Full      No
Grand total is 10100 elements using 81600 bytes
```

可以看出，同样想表示一个 100 × 100 单位阵，如果用稀疏矩阵的方法需要用一个 100 个元素，共 1600 字节的空间 1)，而全部矩阵表示则需要 10000 个元素，共 (80000) 个字节来表示，可见由稀疏矩阵形式来表示会经济得多。

利用 MATLAB 提供的 sparse() 函数也可以将一个普通矩阵转换成稀疏矩阵，该函数的调用方法为

```
B=sparse(A);
```

例如如果 A 矩阵为 A=[1,2,3;4,5,6;7,8,0]，则可以通过下面的命令来获得 A 矩阵的稀疏表示结果

```
>>  B=sparse(A)
B =     (1,1)        1          (2,1)        4
        (3,1)        7          (1,2)        2
        (2,2)        5          (3,2)        8
        (1,3)        3          (2,3)        6
```

如果再采用 whos 命令则可以看出，同样一个矩阵 A，用常规的方法需要 72 个字节，而稀疏表示法需要使用 108 个字节。通过这个例子可以看出，并不是所有的矩阵采用稀疏表示方法都可以节省存储空间，如果一个矩阵的全部元素都不为 0，则采用稀疏矩阵表示方法所用到的元素个数要比常规表示方法多出 50%。

稀疏矩阵也可以通过 MATLAB 提供的 full() 函数的调用转换成常规矩阵的形式。full() 函数的调用方法也是很直观的，用户只需在括号中填入稀疏矩阵的名称，则会自动返回常规矩阵的表示。

更一般地，可以按照下面的调用方法来产生一个稀疏矩阵

```
S = sparse(i, j, s, m, n, nzmax);
```

其中 n，m 分别为最终产生的稀疏矩阵行列数，i，j，s 为一个子矩阵，返回的 S 为最后生成的稀疏矩阵。例如 S=sparse(1:n, 1:n, 1) 将产生一个 n 阶稀疏单位阵，它和 speye(n) 是等效的，最终的结果和 sparse(eye(n)) 也是一致的，所不同的是，用最后一种方法将先暂时地产生一个 $n \times n$ 的常规矩阵，然后才产生单位稀疏矩阵。

再考虑下面的命令 B=sparse(10000, 10000, 1)，这一命令将产生一个 10000×10000 的矩阵且其最后一个元素的值为 1。注意，这时不能使用 full(B) 来将之转换为常规矩阵，因为 B 的常规矩阵表示需要 800MB 的存储空间，而一般的计算机都无法直接表示这样大的矩阵。这样的稀疏矩阵由 MATLAB 输入以后，则用 eig(B) 函数调用可以立即求出原矩阵的特征值来。

---

1) 在 MATLAB 稀疏矩阵的表示方法下，一个元素要占用 16 个字节的空间，这是因为要表示一个矩阵元素，既要用 8 个字节来表示该元素的值，又要用两个 4 字节的整数来表示其行列的值。



MATLAB 提供的内部函数一般都可以直接用于稀疏矩阵的处理，例如由 eig(A) 可以同样得出稀疏矩阵 A 的特征值与特征向量。如果把稀疏矩阵与常规矩阵作运算，例如加法运算，则得出的结果矩阵正常情况下为常规矩阵，但也有特例情况。例如 A 为一个稀疏矩阵，而 B 为常规矩阵，则运算 $A.*B$ 只能比原来的 A 矩阵更稀疏，故这时得出的结果仍为一个稀疏矩阵。

有些矩阵的维数相当大，而往往用户对它的各个元素的具体值不是很感兴趣，只是想对该矩阵的非零元素的分布作一个定性的了解，则可以调用 MATLAB 提供的 spy() 函数将稀疏矩阵的非零元素由图示的方法示意地显示出来，该函数的调用格式为

```
spy(A，标号大小，标号颜色)
```

其中 A 为稀疏矩阵的名称，标号大小为一个数值，用户可以用试探的方式来选择一个合适的大小，颜色标号和 plot() 函数中的类似，如选为 'r' 则表示绘制红色的示意图形。如果 A 是一个 MATLAB 下的稀疏矩阵，则调用 spy(A) 可以绘制出该矩阵非零元素的分布图形。

例 3.19 在有限元计算中，往往要用到大型的稀疏矩阵，例如 MATLAB 4.0 版本中提供了一个数据文件 crack.mat，它包含 22KB 字节，该文件中定义了一个稀疏矩阵，其维数为 136×136，如果由该稀疏矩阵构造一个相同的常规矩阵 A，则再查看整个工作空间将得出如下的结论

```
>>  load crack;   A = full(crack);
>>  whos
      Name        Size       Elements    Bytes    Density   Complex
         A      136 by 136      18496    147968      Full      No
      crack     136 by 136        844     10672    0.0456      No
   Grand total is 19340 elements using 158640 bytes
```

可以看出，该矩阵中有 844 个非零元素，所以稀疏表示形式占用的空间只是常规表示方式的 4.56%，可见对这样的数据使用稀疏表示方式还是很经济的。

如果调用 spy() 函数，则可以得出如图 3-1 所示的非零元素示意图，其中的黑点相对于该位置有非零元素。由此可以容易地定性观察稀疏矩阵的分布情况及特点。

仿照前面介绍的随机元素矩阵，生成具有随机元素的稀疏矩阵可以简单地由 sprand() 或 sprandn() 函数来生成，它们将分别生成满足均匀分布及正态分布的稀疏矩阵。这里所谓的随机分布是指矩阵非零元素的大小满足相应分布，当然非零元素出现的位置也是随机的。以 sprandn() 函数为例，其调用格式为

```
R = sprandn(S)
```

该函数将产生一个和稀疏矩阵 S 一样稀疏度的随机元素稀疏矩阵，且稀疏矩阵的元素满足标准正态分布，而

```
R = sprandn(m,n,D)
```

将产生一个 m 行 n 列，稀疏矩阵密度大约等于 D 的随机元素稀疏矩阵。

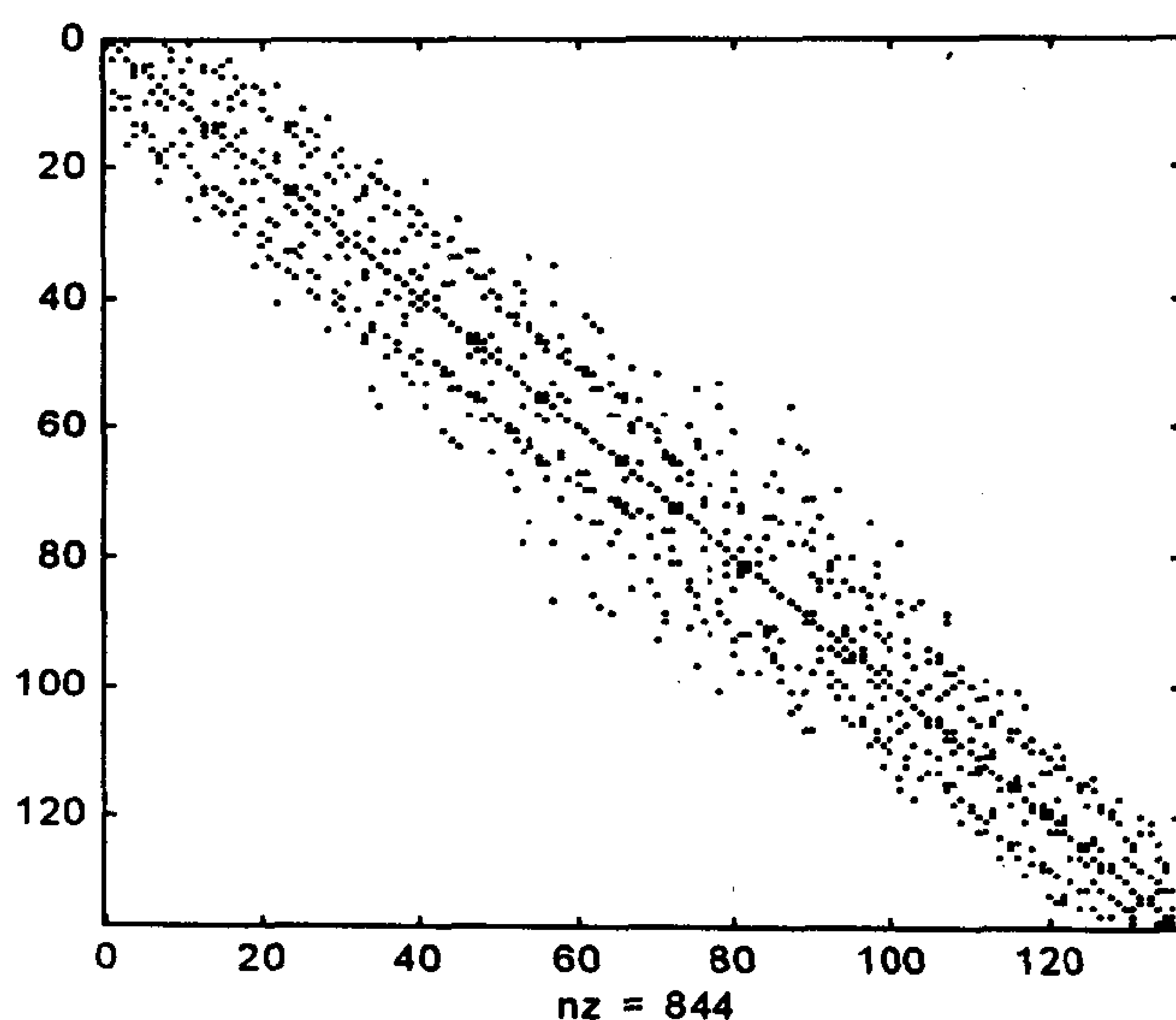
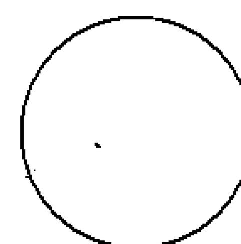


图3-1 稀疏矩阵非零元素示意图

例 3.20 考虑产生两个 100×100 的随机元素矩阵 R 和 S，并预设其稀疏密度为 0.03，然后再显示出该稀疏矩阵的元素分布图形，这样整个过程可以由下面的命令来进行处理

```
>>  S = sprandn(100,100,0.03);  R=sprandn(100,100,0.03);
>>  subplot(121), spy(S)
>>  subplot(122), spy(R)
```



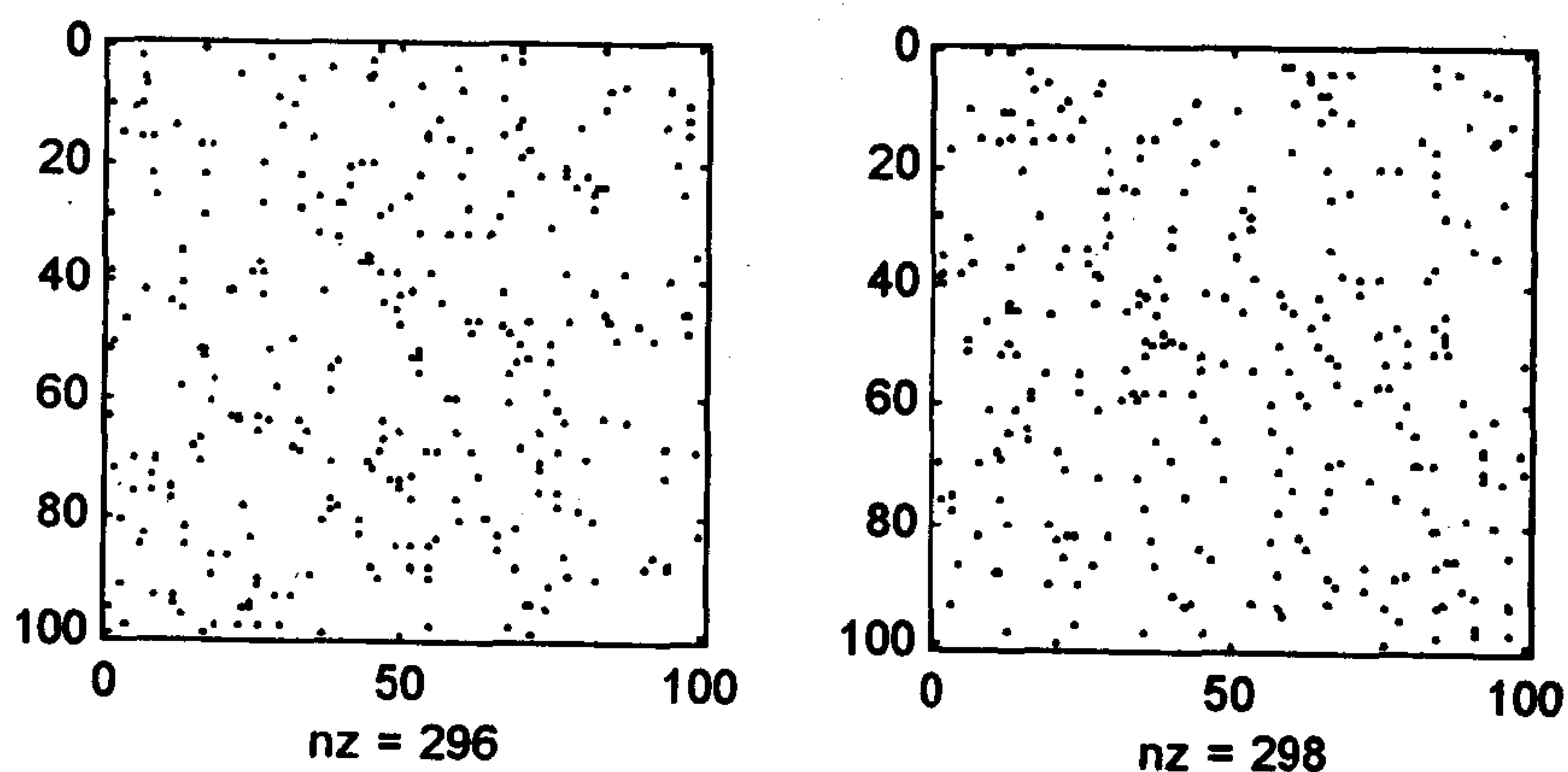


图3-2 随机元素稀疏矩阵分布示意图

这时得出的稀疏矩阵分布图形分别如图 3-2(a), (b) 所示。注意这里显示的两个稀疏矩阵的分布明显是不同的，虽然它们都有几乎相同的稀疏密度。

3.7 矩阵的非线性运算

3.7.1 面向矩阵各个元素的非线性运算

MATLAB 提供了大量函数，允许用户对矩阵进行处理，前面介绍的主要是矩阵的线性变换，在本节中将介绍如何对矩阵进行非线性运算。

事实上，MATLAB 提供了两类函数，其中一类是对矩阵的各个元素进行运算的，而后一类是对整个矩阵进行运算的，前面曾经用到了 sin() 函数，该函数属于第一类，是对矩阵的各个元素单独运算的，而不是对整个矩阵进行运算的。这类常用的 MATLAB 函数在表 3-2 中列出来，它们的调用方法是很显然的，其标准调用格式为

`B = 函数名 (A);`　　如　B=sin(A);

表3-2　面向矩阵元素的非线性函数表

函数名	意　　义	函数名	意　　义
abs()	求模（绝对值）函数	asin(), acos()	反正弦、余弦函数
sqrt()	求平方根函数	log(), log10()	自然和常用对数
exp()	指数函数	real(), imag(), conj()	求实虚部及共轭复数
sin(), cos()	正弦、余弦函数	round(), floor(), ceil()	取整数函数

例 3.21　考虑下面的简单矩阵 A=[1,2,3; 4,5,6; 7,8,0]，调用其中的一些函数其结果在下面给出

```
>> A=[1,2,3; 4,5,6; 7,8,0];
>> exp(A)
ans =   1.0e+003 *
    0.0027     0.0074     0.0201
    0.0546     0.1484     0.4034
    1.0966     2.9810     0.0010
>> log(A)
Warning: Log of zero
ans =        0     0.6931     1.0986
        1.3863     1.6094     1.7918
        1.9459     2.0794       -Inf
>> B = 10*inv(A)
B = -17.7778     8.8889    -1.1111
     15.5556    -7.7778     2.2222
     -1.1111     2.2222    -1.1111
```

```
>> sin(A)
ans =
    0.8415     0.9093     0.1411
   -0.7568    -0.9589    -0.2794
    0.6570     0.9894          0
>> log10(A)
Warning: Log of zero
ans =        0     0.3010     0.4771
        0.6021     0.6990     0.7782
        0.8451     0.9031       -Inf
>> round(B)
ans =    -18      9     -1
          16     -8      2
          -1      2     -1
```

当然这些函数的调用还是很直观的，其中在调用对数函数时给出了错误信息，这是由于对 0 求对数而引起的正常现象。另外这里演示了 round() 函数的基本原理，即对矩

阵的各个元素取不足整数值，floor(), ceil() 及 fix() 等函数的功能也是很显然的，对上面例子调用它们并观察结果，就可以了解其功能了。

MATLAB 还允许调用 fplot() 函数来直接绘制出基本函数的曲线图形，该函数的调用格式为

```
fplot(函数名，绘图区间，样本点数 N)
```

其中函数名为表 3-2 中给出的任意名称，绘图区间为一个 1×2 向量，分别给出绘制图形时自变量的最小值与最大值，样本点数可以缺省，这样将取 N=25，否则用户可以指定任意数值。例如如果用户想绘制出一个周期内的正弦曲线，则可以给出如下的命令

```
fplot('sin',[0,2*pi]);
```

它所起到的作用和下面各个 MATLAB 命令是一致的

```
>>  t=0: 2*pi/25: 2*pi; y=sin(t);
>>  plot(t,y)
```

但这样调用 fplot() 函数有一个最大的好处，就是它不必单独占用存储空间，而后面的命令语句却至少需要生成 t 向量，且调用起来不那么直观。

3.7.2 面向整个矩阵的非线性运算

除了对矩阵的单个元素进行单独计算以外，一般还常常要求对整个矩阵作这样的非线性运算，例如想求出一个矩阵的 e 指数，这就需要特殊的算法来完成了。文献 [8] 中叙述了求解矩阵指数的 19 种不同的方法，每一种方法都有自己的特点及适用范围。在 MATLAB 下提供了 4 个求取矩阵指数的函数：expm(), expm1(), expm2() 和 expm3()，其中的 expm() 为内部函数，它采用 Padé 近似来求取矩阵的指数，而 expm1() 函数是 expm() 函数的 M 函数实现。函数 expm2() 采用 Taylor 级数展开方法来求取矩阵的指数，该方法比较直观，首先对矩阵指数作幂级数展开

$$e^A=\sum_{i=0}^{\infty}\frac{1}{i!}A^i=I+A+\frac{1}{2}A^2+\frac{1}{3!}A^3+\cdots+\frac{1}{m!}A^m+\cdots \tag{3.7.1}$$

若存在一个 m 使得第 $m+1$ 个累加项 $A^{m+1}/(m+1)!$ 的范数足够小时，则认为已经求出了 A 矩阵的指数矩阵，基于该算法的 MATLAB 函数 expm2() 的核心部分清单如下

```
function E = expm2(A)
E = zeros(size(A));
F = eye(size(A));
k = 1;
while norm(E+F-E,1) > 0
   E = E + F;  F = A*F/k;  k = k+1;
end
```

可以看出，这里求取矩阵指数的关键是 while 循环，它在不满足误差要求的时候每一次在结果上累加一个幂级数项，最后得出矩阵的指数来。



expm3() 采用特征值特征向量的方法求出矩阵的指数矩阵，其数学原理如下，首先求出矩阵 A 的特征值 $D=\mathrm{diag}(\gamma_1,\gamma_2,\cdots,\gamma_n)$ 及相应的特征向量矩阵 V, 然后对该对角矩阵求取矩阵指数 (亦即对每个对角矩阵元素求指数)，这时原矩阵 A 的指数矩阵为

$$\mathrm{e}^{A}=V\begin{bmatrix}\mathrm{e}^{\gamma_1} & & \\ & \ddots & \\ & & \mathrm{e}^{\gamma_n}\end{bmatrix}V^{-1} \tag{3.7.2}$$

该算法的 MATLAB 实现为

```
function E = expm3(A)
[V,D] = eig(A);   E = V * diag(exp(diag(D))) / V;
```

这种方法看似简单，但有很大的局限性，它一般要求原矩阵没有重根，否则往往得出的特征向量矩阵趋于奇异，因而得出错误的结果，后面将通过例子来演示这种现象。

例 3.22 考虑下面给出的矩阵 A

$$A=\begin{bmatrix}-2 & 1 & 0 & & \\ 0 & -2 & 1 & \multicolumn{2}{c}{0} \\ 0 & 0 & -2 & & \\ & 0 & & -5 & 1 \\ & & & 0 & -5\end{bmatrix}$$

如果想对此矩阵进行指数运算，则可以获得以下的结果

```
>>  A=[[-2 1 0; 0 -2 1; 0 0 -2], zeros(3,2); zeros(2,3) [-5 1; 0 -5]];
>>  expm(A)
ans =   0.1353     0.1353     0.0677          0          0
             0     0.1353     0.1353          0          0
             0          0     0.1353          0          0
             0          0          0     0.0067     0.0067
             0          0          0          0     0.0067
>>  logm(ans)
Warning: LOGM appears inaccurate.  esterr = 4.591e-007
ans =
  -2.0000+0.0000i   1.0000-0.0000i  0.0000-0.0000i  0.0000-0.0000i   0.0000+0.0000i
   0.0000-0.0000i  -2.0000-0.0000i  1.0000-0.0000i  0.0000+0.0000i   0.0000+0.0000i
   0.0000+0.0000i   0.0000-0.0000i -2.0000+0.0000i  0.0000-0.0000i   0.0000-0.0000i
   0.0000-0.0000i   0.0000-0.0000i  0.0000+0.0000i -5.0000+0.0000i   1.0000+0.0000i
   0.0000+0.0000i   0.0000+0.0000i  0.0000-0.0000i  0.0000+0.0000i  -5.0000-0.0000i
>>  real(ans)
ans = -2.0000     1.0000     0.0000     0.0000     0.0000
       0.0000    -2.0000     1.0000     0.0000     0.0000
       0.0000     0.0000    -2.0000     0.0000     0.0000
       0.0000     0.0000     0.0000    -5.0000     1.0000
       0.0000     0.0000     0.0000     0.0000    -5.0000
>>  norm(ans-A)
ans =   1.1280e-006
```




在这里对 e^A 的对数运算不是很精确，且伴随产生了微小的虚数分量，如果取所得出结果的实部，则表面上可以还原出原来的 A 矩阵，但如果进一步进行误差分析，则可以发现这样的还原结果误差为 1.1280×10^{-6}，该误差虽然在有些软件下被认为是合理的，但在 MATLAB 标准下将被认为太高，所以给出错误信息提示。事实上，原来的矩阵 A 为一个标准的 Jordan 矩阵，所以如果对两个 Jordan 子矩阵作指数运算，也可以得出该矩阵的指数矩阵

```
>> B=[expm(A(1:3,1:3)), zeros(3,2); zeros(2,3), expm(A(4:5,4:5))]
B =  0.1353    0.1353    0.0677         0         0
          0    0.1353    0.1353         0         0
          0         0    0.1353         0         0
          0         0         0    0.0067    0.0067
          0         0         0         0    0.0067
```

可见两种方法得出的指数矩阵是相同的。如果采用 expm2() 函数来求取矩阵指数，则可以发现要经过幂级数的 41 步累加，且其累加项的范数可达 9.0548×10^{-20}，显见这一函数可以以足够的精度得出正确的结果。对同样一个矩阵采用 expm3() 来求取矩阵指数，则可以得出

```
>> B=expm3(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 6.400000e-039
B =  0.1353         0         0         0         0
          0    0.1353  -35.7010         0         0
          0         0    0.1353         0         0
          0         0         0    0.0067         0
          0         0         0         0    0.0067
```

其中给出了警告信息，通知用户得出的特征向量矩阵为坏条件矩阵 (条件数的倒数达 10^{-39})，分析一下原矩阵的特征值

```
>> eig(A)'
ans =   -2    -2    -2    -5    -5
```

可以看出，该矩阵有两组重根：-2 (3 重) 和 -5 (2 重)，这将使得得出的特征向量矩阵出现奇异现象，因而导致得出错误的矩阵指数。所以在使用 expm3() 函数时需要引起足够的重视，以免得出错误的结果。

除了对整个矩阵求取矩阵指数之外，MATLAB 还允许对矩阵进行其它非线性变换，其中常用的函数还有

logm() (矩阵求对数) sqrtm() (矩阵求平方根) 和 funm() (矩阵求任意函数)

可以看出，这里的函数名很有特点，每个函数名在标准函数名的后面加了一个后缀 m，表示对矩阵而不是对矩阵元素进行运算。这里的 funm() 函数可以求出矩阵的任意函数，其调用方法为

funm(矩阵名,' 函数名')



其中函数名应该由单引号括起来，例如如果想求出矩阵 A 的正弦矩阵，则可以使用如下的命令 B = funm(A, 'sin')。值得指出的是，这里给出的矩阵函数运算是基于矩阵特征值特征向量而完成的，故在一些特殊 (但很常见) 的场合下仍将出现错误。

重新考虑前面的例子，如果想对其中的 A 矩阵作正弦运算，则将得出如下的错误结论

```
>>  funm(A,'sin')
WARNING: Result from FUNM may be inaccurate. esterr = 1
ans = -0.9093         0         0         0         0
            0   -0.9093         0         0         0
            0         0   -0.9093         0         0
            0         0         0    0.9589         0
            0         0         0         0    0.9589
```

事实上矩阵的非线性函数运算可以通过幂级数的方法简单地求出，例如正弦函数可以由下面的幂级数展开式求出

$$\sin(A)=\sum_{i=0}^{\infty}(-1)^i\frac{A^{2i+1}}{(2i+1)!}=A-\frac{1}{3!}A^3+\frac{1}{5!}A^5+\cdots \tag{3.7.3}$$

仿照 expm2() 函数，正弦函数幂级数展开的 MATLAB 实现为

```
E = zeros(size(A));
F = A;
k = 1;
while norm(E+F-E,1) > 0
   E = E + F;  F = -A^2*F/((k+2)*(k+1)); k = k+2;
end
```

由上面的程序可以看出，看起来比较复杂的矩阵幂级数展开程序可以由几条 MATLAB 语句容易地编写出来。在习题中给出了余弦及反正弦函数的幂级数展开公式，用户可以根据这些公式，并仿照前面给出的程序段很简洁地写出相应的程序来。例如利用前面给出的程序段可以在 39 步幂级数累加之后容易地求出原矩阵 A 的正弦矩阵

```
>>  k, E
k =   39
E = -0.9093   -0.4161    0.4546         0         0
          0   -0.9093   -0.4161         0         0
          0         0   -0.9093         0         0
          0         0         0    0.9589    0.2837
          0         0         0         0    0.9589
```

对上面的结果矩阵再作反正弦运算，不难得出这样的结论，这种运算可以还原出原来的矩阵 A，而这样的结果光靠 MATLAB 提供的 funm() 函数是不可能得出的，所以在使用 funm() 函数时应该格外注意，如果确实不能得出正确的结果，则建议采用幂级数的方法编写程序来直接求出。

3.8 数值分析的其它方法及 MATLAB 实现

前面给出了各种数值线性代数的基本算法及 MATLAB 实现，这里将讨论其它的数值分析方法及 MATLAB 实现，包括数据处理、数值微积分、非线性方程求解和最优化以及常微分方程的数值解法等内容。严格说来，本节的内容并不属于数值线性代数的范畴，但为了叙述方便起见，在这里一起介绍这些内容并不会给读者造成太多的误解，而相反地会更进一步理解数值分析基本内容，并学会利用 MATLAB 提供的功能来迅速地解决遇到的问题。

3.8.1 数据处理的方法及 MATLAB 实现

如果给定一组数据 $\{x_i\}, i = 1, 2, \cdots, L$ ，则可以用 MATLAB 将这些数据用一个向量表示出来，其具体方法当然是

```
x=[x1, x2, ..., xL];
```

这时就可以用 MATLAB 提供的各种命令或函数对这样的数据进行处理了。例如如果想求出这样一组数据的最大值和最小值，则可以分别采用下面的函数来求出

`[xM, i]=max(x);` 或 `[xm, i]=min(x);`

其中返回的 xM 及 xm 分别为向量 x 的最大值或最小值，而 i 为最大值或最小值所在的位置，当然这两个函数均可以值返回一个参数，而不返回 i 。如果给出的 x 不是向量而是矩阵，则采用 min() 和 max() 函数得出的结果将不是数值，而是一个向量。它的含义是得出由每一列构成的向量的最小值或最大值所构成的行向量。MATLAB 还提供了对给定向量的大小进行排序的函数 sort()，其调用格式和 min() 的几乎是完全一致，调用了此函数之后，就可以将向量的值按照从小到大的顺序进行排列。

实验数据或随机数据也可以由 MATLAB 给出的数据处理函数容易地进行，比如函数 mean() 可以立即求出向量的均值，而 std() 可以求出向量的标准方差，而 median() 可以求出向量的中间值。MATLAB 提供的 cov() 函数可以对矩阵进行处理，得出数据的协方差。

例 3.23 利用 MATLAB 提供的函数首先可以容易地生成满足正态分布的伪随机数矩阵

```
>>  S=randn(10000,5);
```

上面的语句可以生成一个 10000 × 5 的伪随机数矩阵，它们各列之间的协方差可以由下面的语句求出

```
>>  V=cov(S)
V =  1.0077    0.0116    0.0065   -0.0079   -0.0032
     0.0116    1.0226    0.0036    0.0041   -0.0029
     0.0065    0.0036    1.0167    0.0127   -0.0097
    -0.0079    0.0041    0.0127    0.9860   -0.0037
    -0.0032   -0.0029   -0.0097   -0.0037    0.9980
```




这样就可以容易地产生一个 5 × 5 的协方差矩阵 (理论上是单位矩阵)。调用其它 MATLAB 函数可以针对随机矩阵的各列得出下面的结果

```
>> mean(S)
ans =  -0.0118    0.0050    0.0037   -0.0023    0.0072
>> std(S)
ans =   1.0038    1.0112    1.0083    0.9930    0.9990
>> median(S)
ans =  -0.0109    0.0006    0.0200   -0.0026    0.0170
```

由满足标准正态分布的随机数性质可以显然地看出，上面的结果是正确的，例如产生的均值都很小，而标准方差接近于 1 。

MATLAB 还提供了其它数据处理的方法，如样条函数插值 spline() 及快速 Fourier 变换函数 fft() 等，这里就不再进一步介绍了，有兴趣的读者可以由联机帮助得出有关的信息。

3.8.2 数值积分方法及 MATLAB 实现

在控制系统 CAD 或其它计算中，经常要求解一个函数的定积分

$$I = \int_a^b f(x)\mathrm{d}x \tag{3.8.1}$$

在被积函数 $f(x)$ 相当复杂时，一般很难采用解析的方法求出定积分的值来，而往往要采用数值方法来求解。在数值分析课程中我们知道，求解定积分的数值方法是多种多样的，如简单的梯形法、Simpson 法、Romberg 法等都是经常采用的方法。它们的基本思想都是将整个积分空间 $[a,b]$ 分割成若干个子空间 $[x_i, x_{i+1}]$, $i = 1, 2, \cdots, N$, 其中 $x_1 = a$, $x_{N+1} = b$ 。这样整个积分问题就分解为下面的求和形式

$$\int_a^b f(x)\mathrm{d}x = \sum_{i=1}^{N} \int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x \tag{3.8.2}$$

而在每一个小的子空间上都可以近似的求解出来，例如可以采用下面给出的 Simpson 方法来求解出 $[x_i, x_{i+1}]$ 上的积分近似值

$$\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x \simeq \frac{h_i}{12}\left[f(x_i)+4f\left(x_i+\frac{h_i}{4}\right)+2f\left(x_i+\frac{h_i}{2}\right)+4f\left(x_i+\frac{3h_i}{4}\right)+f(x_i+h_i)\right] \tag{3.8.3}$$

式中 $h_i = x_{i+1} - x_i$ 。MATLAB 基于此算法，采用自适应变步长方法给出了 quad() 函数来求取定积分，该函数的调用格式为

```
y=quad(函数名 ,  a, b, tol);
```

其中 a, b 分别为定积分的上下限， tol 为变步长积分用的误差限，如果用户不给出误差限，则将自动地假定 tol=1e-3 。在调用 quad() 函数时，用户首先应该编写一个描述 $f(\cdot)$ 的函数，其格式为

```
y= 函数名 (x);
```

例 3.24　试求出下面的定积分

$$\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-x^2/2}dx$$

这一无穷定积分可以由有穷的积分来近似，一般情况下，选择积分的上下限为 ±15 就能保证相当的精度。由给出的被积函数可以简单地写出下面的函数

```
function y=myerrf(x)
global kk; kk=kk+1;
y=1/sqrt(2*pi)*exp(-x.^2/2);
```

这里设置了一个全局变量 kk, 其作用是累计 myerrf() 函数调用的次数，一般情况下没有必要在函数中包含这一语句。假定这时可以通过下面的 MATLAB 语句求出所需函数的定积分

```
>>  global kk; kk=0;
>>  format long; y=quad('myerrf',-15,15), kk
y = 1.00000007062479
kk=   861
```

如果将精度选择得过高，则可能出现 Recursion level limit reached in quad. Singularity likely 字样的提示，其实这一算法的精度不是很高，且需要调用 861 次 myerrf() 函数。

文献 [2] 给出了一种高精度的算法，该算法利用了插值运算来更精确更快速地求出所需要的定积分。该算法又称为 Newton Cotes 方法。例如，在 [0, 1] 区间上的积分可以近似地表示成

$$\int_0^1 f(x)dx \simeq \sum_{k=0}^{8} w_k f\left(\frac{k}{8}\right) \tag{3.8.4}$$

其中加权系数 w_k 为

$$w_0 = \frac{989}{28350},\ w_1 = \frac{2944}{14175},\ w_2 = -\frac{4644}{14175},\ w_3 = \frac{5248}{14175}, w_4 = -\frac{454}{2835} \tag{3.8.5}$$

且 $w_{8,7,6,5} = w_{0,1,2,3}$，这些加权系数满足

$$\sum_{k=0}^{8} w_k = 1,\quad \sum_{k=0}^{8} | w_k |= 1.4512 \tag{3.8.6}$$

在这一算法下的定积分求值的 MATLAB 实现在函数 quad8() 中给出，该函数的调用格式为

```
y=quad8(函数名 ,  a, b, tol);
```

该函数的调用格式和 quad() 几乎一致，该函数下 tol 的默认值为 10^{-6}。该算法可以更精确地求出积分的值，且一般情况下函数调用的步数明显小于 quad()，故而保证能以更高的效率求解出所需的定积分值。

对于上面同样的积分使用 quad8() 来求解所得出的结果如下



```
>> kk=0; y=quad8('myerrf',-15,15); kk
y = 0.99999999999999
kk=    96
```

该函数仅调用了 96 次 myerrf() 函数，因而其效率要明显高于 quad() 函数，且精度也高得多(理论值为 1)，且不会出现像 quad() 函数调用时那样的错误信息提示。

3.8.3 非线性方程求解及最优化

由前面的介绍可以看出，线性代数方程和多项式方程均可以在 MATLAB 下通过一个函数的调用而直接求出，并且求出的结果可以保证很高的精度。非线性方程组的求解方法比较复杂，前面通过例子介绍了 Newton 迭代法及其 MATLAB 实现，通过该算法当然可以求出非线性方程的一个实根，但该方法需要已知原方程的导数，而在实际运算中这一条件有时是不能满足的，所以又出现了弦切法等其它解法。在 MATLAB 中给出了一个函数 fzero(), 可以用来求解含有一元方程的解。

在 MATLAB 的早期版本中还提供了求解多元方程的求解函数 fsolve(), 该函数的基本调用格式为

```
x=fsolve(函数名，初值);
```

其中 x 为返回的解，函数名是由引号括起来的，在该函数中对非线性方程组有一个一般的描述，而初值为求根过程的起始点。但在 MATLAB 4.0 中没有提供此函数，如果需要也可以将早期版本的 fsolve.m 文件及其组成部分的 ne*.m 文件复制到 MATLAB 4.0 的 funfun 目录下来执行。

例 3.25 考虑下面的三元方程

$$\sin x + y^2 + \ln z - 7 = 0, \quad 3x + 2^y - z^3 + 1 = 0, \quad x + y + z - 5 = 0$$

可以编写出一个解三元方程的函数 my3deq(), 其内容为

```
function q = my2deq(p)
global kk; kk=kk+1;
q(1)=sin(p(1))+p(2)^2+log(p(3))-7;
q(2)=3*p(1)+2^p(2)-p(3)^3+1;
q(3)=p(1)+p(2)+p(3)-5;
```

其中 kk 仍为表示调用此函数次数的全局变量。这样就可以在给定的初值 $x_0=1, y_0=1, z_0=1$ 下调用 fsolve() 函数直接求出方程的根

```
>> global kk; kk=0; x = fsolve('my3deq',[1 1 1]'), kk
x =  0.5991
     2.3959
     2.0050
kk =   14
```

MATLAB 提供了两个基于文献 [9] 中给出的单纯形算法求解最优化的函数 fmin() 和 fmins(), 它们分别对应于单变量和多变量的最优化问题的求解，其调用格式是接近的

x=fmin(函数名，初值，选项)　或　x=fmins(函数名，初值)

其中函数名的定义和其它函数是一致的，而初值的大小往往能决定最后解的精度和收敛速度。选项是由一些控制变量构成的向量，比如它的第一个分量不为 0 表示在求解时显示整个动态过程 (其默认值为 0)，第二个分量表示求解的精度 (默认值为 1e-4), 用户可以指定这些参数来控制求解的条件。

例 3.26　考虑下面的 Rosenbrock 最小化函数

$$f(x_1,x_2)=100(x_2-x_1^2)^2+(1-x_1)^2$$

该函数的 MATLAB 表示为

```
function y=myfun(x)
global kk;  kk=kk+1;
y=100*(x(2)-x(1)*x(1))^2+(1-x(1))^2;
```

编写了这样的函数后，就可以由下面的命令来求解最优化问题了

```
>>  global kk; kk=0;
>>  x=fmins('myfun',[-1.2, 1]), kk
x= 1.00000414419788    1.00001138812015
kk =    90
>>  myfun(x)
ans=  9.7800e-010
```

求解这样的问题需要调用 90 次 myfun() 函数，而最终得出的函数值为 10^{-10} 级。如果指定求解的精度为 1e-10, 则可以得出

```
>>  global kk; kk=0;
>>  x=fmins('myfun',[-1.2, 1],[0,1e-10]), kk
x=  0.99999999999105    0.99999999997975
kk =    175
>>  myfun(x)
ans = 6.309077360658436e-022
```

可见采用此算法一般可以得出较满意的结果。求解精度为 1e-15 和 1e-20 分别要调用 myfun() 函数 253 和 401 次，而在精度选择为 10^{-20} 时函数的值为 5.9658×10^{-30}。

除了标准的最优化函数之外，还存在两个版本的最优化工具箱 (Optimisation Toolbox)，这两个版本分别为英国学者 Peter Fleming 及美国的 Andrew Grace 所编写的，在这里就不多加介绍了，感兴趣的读者可以参考有关资料。



3.8.4 微分方程初值问题的数值解法

微分方程初值问题的数值解法实际上是系统数值仿真的基础。假设系统的常微分方程满足

$$\dot{x}_i = f_i(\mathbf{x}, t) \tag{3.8.7}$$

其中 $x_i, i = 1, 2, \cdots, n$ 为状态变量，且 $\mathbf{x} = [x_1, \cdots, x_n]$，而 $f_i(\cdot)$ 为任意非线性函数且 $\mathbf{f}(\cdot) = [f_1(\cdot), \cdots, f_n(\cdot)]$， t 为时间变量，这样就可以采用数值方法在初值 $x(0)$ 下来求解常微分方程组了。求解常微分方程组的数值方法是多种多样的，如常用的 Runge-Kutta 方法、 Adams 线性多步法、预报校正法、 Gear 法等，对于一般常微分方程的数值解法在文献 [10] 中给出了较客观的比较。德国学者 Felhberg 对传统的 Runge-Kutta 方法进行了改进，在每一个计算步长内对 $f_i(\cdot)$ 函数进行六次求值，以保证更高的精度和数值稳定性，假设当前的步长为 h_k，则定义了下面的 6 个 k_i 变量

$$k_i = h_k \mathbf{f}\left(\mathbf{x}_k + \sum_{j=1}^{i-1} \beta_{ij} k_j, t_k + \alpha_i h_k\right), \quad i = 1, 2, \cdots, 6 \tag{3.8.8}$$

式中 t_k 为当前计算时刻，而中间参数 α_i, β_{ij} 及其它参数由表 3-3 给出。这时下一步的

表 3-3 四阶 / 五阶 RKF 算法系数表

α_i	β_{ij}					γ_i	γ_i^*
0						16/135	25/216
1/4	1/4					0	0
3/8	3/32	9/32				6656/12825	1408/2565
12/13	1932/2197	-7200/2197	7296/2197			28561/56430	2197/4104
1	439/216	-8	3680/513	-845/4104		-9/50	-1/5
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	2/55	0

状态变量可以由下式求出

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \sum_{i=1}^{6} \gamma_i \boldsymbol{k}_i \tag{3.8.9}$$

这一方法又称为四阶 / 五阶 Runge-Kutta-Felhberg (RKF) 方法。当然直接采用这一方法则为定步长的方法，而定步长的方法在实际中是很不实用的，因为如果步长选得过小，可以保证较高的精度(但也会增加舍入误差)，但计算的时间将比步长大时多花费很多，而步长选择大时计算速度快，但要牺牲一定的精度。在实际问题中往往希望在一些情况下(如解变化很快时)采用较小的步长，而在另一些情况下(如解的变化很缓慢时)采用较大的步长，这样做既可以保证较高的精度，又可以保证较高的速度。这种能自动变换步长的方法又称为自适应变步长方法。在这里介绍的 RKF 方法中给出了变步长的解法，并引入了 $\sum_{i=1}^{6}(\gamma_i - \gamma_i^*)k_i$ 误差量来控制步长的大小。

MATLAB 提供了两个常微分方程求解的函数 ode23() 和 ode45()。这两个函数分别采用了二阶 / 三阶的 RKF 方法和四阶 / 五阶 RKF 方法，并采用自适应变步长的求解方法，即当解的变化较慢时采用较大的计算步长，从而使得计算速度很快，当方程的解变化得较快时，积分步长会自动地变小，从而使得计算的精度很高。这两个函数的调用格式分别为

```
[t, x]=ode23(系统函数名 ,  t0, tf, x0, tol, trace)
[x, t]=ode45(系统函数名 ,  t0, tf, x0, tol, trace)
```

这里所用到的系统函数名为描述系统状态方程的 M 函数的名称，该函数名应该用引号括起来。 t0 和 tf 分别为用户指定的起始和终止仿真时间， x0 为系统的初始状态变量的值。有了这些参数，就可以调用 ode23() 和 ode45() 两个函数对系统直接进行仿真了。为了限定变步长仿真的精度，用户还可以有选择地添加一个附加参数 tol，由它来指定变步长仿真中的期望的解的精度，如果不对之进行指定，则默认的误差限分别为：对 ode23() 函数取 10^{-3}，对 ode45() 函数取 10^{-6}。该函数调用中最后的控制参数 trace 为一个输出形式控制变量，如果 trace 不为 0，则会将仿真的中间结果逐步地由屏幕显示出来，否则将不显示中间结果 (这也是该函数的默认状态)。不论是否显示中间的仿真结果，此函数都将返回两个变量 t 和 x，其中 t 为仿真的时间变量，因为采用了变步长的仿真算法，所以得出的 t 向量并不一定是等间隔的，另一个变量 x 为状态变量在各个时刻所组成的列向量所构成的矩阵的转置，有了 t 和 x 这样的变量，在仿真结束后当然可以采用 plot(t,x) 来绘制出仿真的结果曲线。

值得指出的是，这里要用到的系统函数名的编写格式是固定的，如果其格式没有按照要求去编写，则将得出错误的仿真结果。系统函数的编写格式为

```
function xdot= 函数名 (t,x)
```

其中 t 为时间变量， x 为状态变量，而 xdot 为状态变量的导数。可见如果想编写这样的函数，首先必须已知原系统的状态方程模型。下面将通过例子来说明系统函数的编写及微分方程求解函数的使用。

例 3.27　考虑著名的 Van der Pol 方程的数值仿真

$$\ddot{x}+(x^2-1)\dot{x}+x=0$$

仿照第一章中关于 ACSL 的实例，选择状态变量 $x_2=x$, $x_1=\dot{x}$, 则可以列写出如下的状态方程模型

$$\dot{x}_2=x_1,\ \ \dot{x}_1=x_1(1-x_2^2)-x_2$$

基于该状态方程可以编写出一个 M 函数来描述此系统

```
function xdot=vdpol(t,x)
xdot(1)=x(1)*(1-x(2)^2)-x(2); xdot(2)=x(1);
```

当然状态方程表现形式并不是唯一的，例如前面的描述还可以表示成 xdot=[x(1)*(1-x(2)^2)-x(2); x(1)] 或 xdot=[1-x(2)^2, -1; 1, 0]*x。



有了系统的状态方程的 M 函数之后，就可以通过 ode23() 或 ode45() 来进行数字仿真了。例如如果选定初始值为 x0=[0,0.25]，并在 $0 \le t \le 20$ 的时间范围内进行仿真，则可以得出

```
>>  t0=0; tf=20; tol=1e-6; x0=[0; 0.25]; trace=1;
>>  [t,x]=ode45('vdpol',t0,tf,x0,tol,trace);
         0    0.1563          0    0.2500
    0.1563    0.1563    -0.0419    0.2468
    0.4102    0.2540    -0.1217    0.2263
    0.6627    0.2525    -0.2145    0.1841
    0.9081    0.2454    -0.3155    0.1193
    1.1455    0.2373    -0.4207    0.0321
    1.3731    0.2277    -0.5235   -0.0755
    1.5914    0.2182    -0.6158   -0.2000
    1.8056    0.2143    -0.6880   -0.3401
    2.0383    0.2327    -0.7252   -0.5056
     ...        ...        ...        ...
```

这里除了正常返回 t 和 x 之外，还将同时显示中间的结果。这里对显示的格式略加调整[1]就可以得出上面的较简洁的显示，其中第一列参数为仿真时刻，第二列参数为仿真步长，而第三列及以后的数据表示各个状态变量在每个时刻的数值。在此函数的调用中一共对 vdpol() 函数调用了 1056 次，并得出了 159 组仿真数据。同样的精度下调用 ode23() 函数来求解时共需调用 4635 次 vdpol() 函数，并返回 1533 组仿真数据。可见 ode45() 的效率要远远高于 ode23()。如果在调用此函数时将 trace 设置成 0，则将不显示中间结果。得出了仿真结果之后，则可以由下面的 MATLAB 命令

```
>>  subplot(121), plot(t,x)
>>  subplot(122), plot(x(:,1), x(:,2))
```

就可以绘制出系统的响应曲线及相平面图形，这些图形分别如图 3-3(a), (b) 所示。

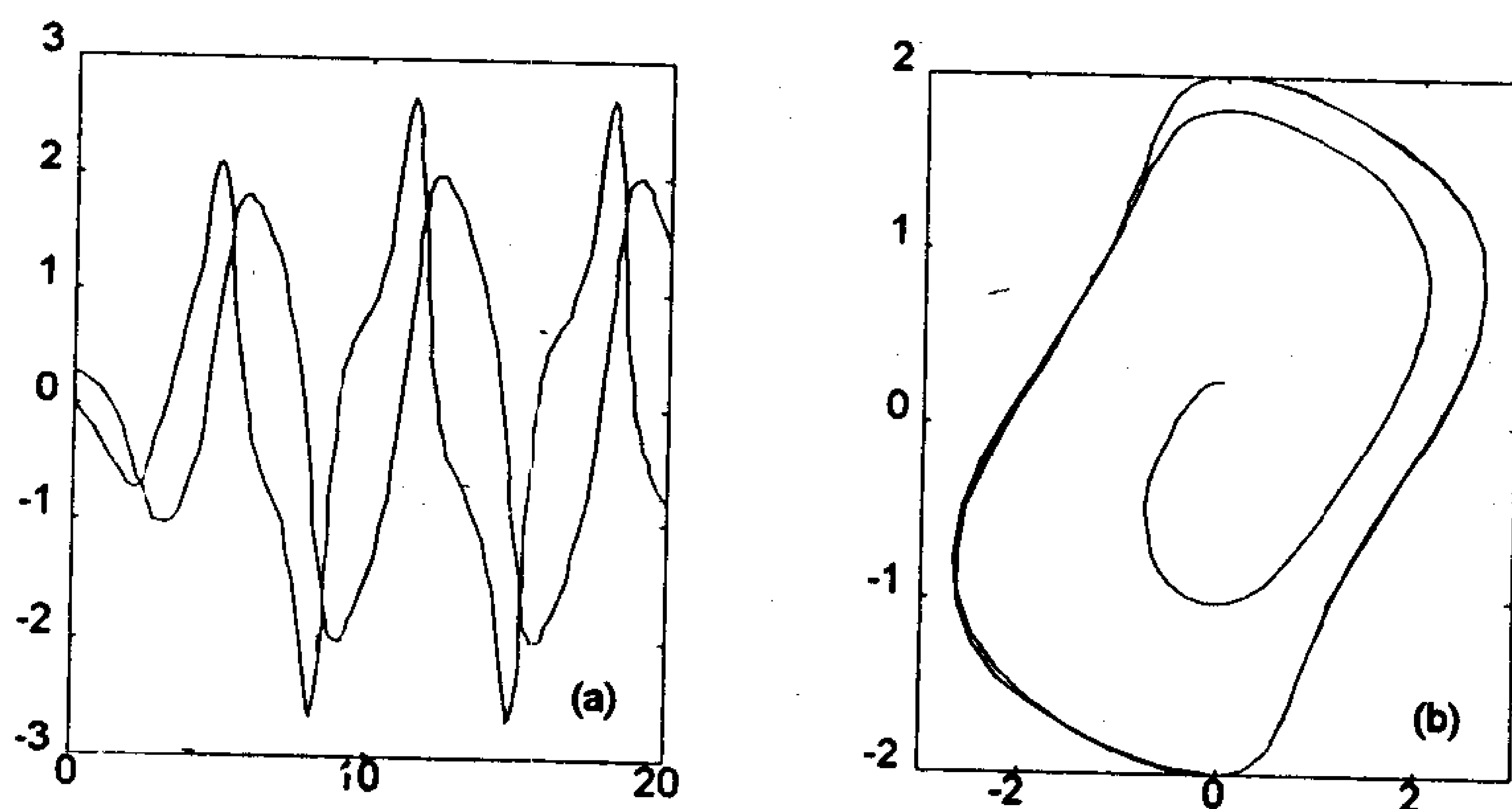


图 3-3 Van der Pol 方程的时间响应与相平面图

此外，在第二章的末尾曾使用过名为 ode23p() 的函数，该函数的功能和 ode23() 几乎是一致的，所不同的是它将在仿真过程中逐点地绘制出响应曲线来。注意，在使用该函数之前应该首先设定绘图范围。

[1]将原函数中的 home, t, h, y 字样改变成 disp([t, h, y']) 即可。

其实，对常微分方程来说，初值问题的数值解法 (或称仿真算法) 是多种多样的，除了这里介绍的 Runge-Kutta 法之外，比较常用的还有 Euler 法、预报校正法、Adams 法、Rosenbrock 法、 Gear 法等，它们的侧重应用范围是不一致的，其中一些方法侧重于一般问题的仿真，而另一些方法侧重病态方程 (stiff equation) 的仿真。在带有 SIMULINK 的 MATLAB 环境中，以内部函数的方式实现了其中一些仿真算法，这些算法的调用在介绍 SIMULINK 时再加以详细介绍。

习　　题

1) 构造一个 11 阶的 Hilbert 矩阵，并得出该矩阵的条件数，再用两种方法求取该矩阵的逆，其中一种可以是直接求逆法，另一种可以调用 MATLAB 的 Hilbert 逆矩阵求解函数 `invhilb()`，然后比较两种方法得出的结果，并验证得出的逆矩阵是否真的满足逆矩阵的条件。

2) 编写一个可以生成式 (3.1.6) 中 Hankel 矩阵的函数 `newhank()`，使其调用格式为 `H=newhank(C)`，其中 `C` 为给定向量。

3) 对下面给出的各个矩阵求取各种参数，如矩阵的行列式、迹、秩、特征多项式、范数等

$$A=\begin{bmatrix}7.5&3.5&0&0\\8&33&4.1&0\\0&9&103&-1.5\\0&0&3.7&19.3\end{bmatrix},\quad B=\begin{bmatrix}5&7&6&5\\7&10&8&7\\6&8&10&9\\5&7&9&10\end{bmatrix}$$

$$C=\begin{bmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\\13&14&15&16\end{bmatrix},\quad D=\begin{bmatrix}3&-3&-2&4\\5&-5&1&8\\11&8&5&-7\\5&-1&-3&-1\end{bmatrix}$$

4) 对上面的各个矩阵进行三角分解和奇异值分解，并验证所得出的结果。

5) 求出上面各个矩阵的特征值与特征向量，并进行验证。

6) 对前面的对称矩阵 B 作 Cholesky 分解，并验证结果。

7) 求解下面的线性代数方程

$$\text{(a)}\ \begin{bmatrix}7&2&1&-2\\9&15&3&-2\\-2&-2&11&5\\1&3&2&13\end{bmatrix}X=\begin{bmatrix}4\\7\\-1\\0\end{bmatrix},\quad \text{(b)}\ \begin{bmatrix}1&3&2&13\\7&2&1&-2\\9&15&3&-2\\-2&-2&11&5\end{bmatrix}X=\begin{bmatrix}9&0\\6&4\\11&7\\-2&-1\end{bmatrix}$$

并验证得出的解真正满足原方程。

8) 对下面给出的复数矩阵进行分析，分别求出它的矩阵参数、逆矩阵、特征值与特征向量，并验证结果

$$A=\begin{bmatrix}0.2368&0.2471&0.2568&1.2671\\1.1161&0.1254&0.1397&0.1490\\0.1582&1.1675&0.1768&0.1871\\0.1968&0.2071&0.2168&0.2271\end{bmatrix}+\begin{bmatrix}0.1345&0.1768&0.1852&1.1161\\1.2671&0.2017&0.7024&0.2721\\-0.2836&-1.1967&0.3558&-0.2078\\0.3536&-1.2345&2.1185&0.4773\end{bmatrix}\mathrm{i}$$

9) 如果令输入为一个方阵 A 和一个列向量 c，编写一个函数 `lyap0()`，使其调用格式为

```
X=lyap0(A,c)
```

其中 X 为下面的 Lyapunov 方程的解

$$AX+XA^T=-c^Tc$$

并通过一个例子来验证得出的解确实满足该方程。

10) 下面是一个常用的旋转变换矩阵

$$P=\begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \cos(\theta_i) & -\sin(\theta_i) & \leftarrow \text{第 i 行} & \\ & & & \sin(\theta_i) & \cos(\theta_i) & \leftarrow \text{第 i+1 行} & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix}$$

试用稀疏矩阵的形式来表示它，并在占用的存储空间上和原矩阵相比较。

11) 求出下面给出的矩阵的秩和 Moore-Penrose 广义逆矩阵，并验证它们是否满足 Moore-Penrose 逆矩阵的条件

$$A=\begin{bmatrix} 2 & 2 & 3 & 1 \\ 2 & 2 & 3 & 1 \\ 4 & 4 & 6 & 2 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 3 \end{bmatrix},\quad B=\begin{bmatrix} 4 & 1 & 2 & 0 \\ 1 & 1 & 5 & 15 \\ 3 & 1 & 3 & 5 \end{bmatrix}$$

12) 利用 MATLAB 求出下面矩阵 A 的矩阵指数 e^A, 矩阵对数 $\ln(A)$

$$A=\begin{bmatrix} 5 & 2 & 1 & 0 \\ 0 & 4 & 6 & 0 \\ 0 & -3 & -5 & 0 \\ 0 & -3 & -6 & -1 \end{bmatrix}$$

并对得出的矩阵指数和对数分别进行反变换，再与原来的矩阵 A 进行比较。

13) 对余弦函数、反正弦函数及对数函数分别编写求矩阵变换的 MATLAB 程序段，并利用编写的反正弦程序段对例子中的正弦结果进行检验。这里涉及的各个函数的幂级数展开公式如下：

(a) $\cos(A)=I-\dfrac{1}{2!}A^2+\dfrac{1}{4!}A^4-\dfrac{1}{6!}A^6+\cdots+\dfrac{(-1)^n}{(2n)!}A^{2n}+\cdots$

(b) $\arcsin(A)=A+\dfrac{1}{2\cdot 3}A^3+\dfrac{1\cdot 3}{2\cdot 4\cdot 5}A^5+\dfrac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 7}A^7+\cdots+\dfrac{(2n)!}{2^{2n}(n!)^2(2n+1)}A^{2n+1}+\cdots$

(c) $\ln(A)=-I+A-\dfrac{1}{2}(A-I)^2+\dfrac{1}{3}(A-I)^3-(A-I)^4+\cdots+\dfrac{(-1)^{n+1}}{n}(A-I)^n+\cdots$

(提示：用户当然可以将 (c) 中的方法得出的结果和 `logm()` 进行精度比较。)

14) 求下面给出的定积分的值

$$I_1=\int_0^2\frac{\sin x}{x}\mathrm{d}x,\quad I_2=\int_0^1\left[\frac{1}{(x-0.3)^2+0.01}-\frac{1}{(x-0.9)^2+0.04}-6\right]\mathrm{d}x$$

$$I_3=\int_0^4 f_1(x)\mathrm{d}x,\ \text{其中}\ f_1(x)=\begin{cases} e^{x^2}, & 0\leq x\leq 2 \\ \dfrac{1}{4-\sin(16\pi x)}, & 2<x\leq 4 \end{cases}$$

15) 已知 Apollo 的运动轨迹 (x,y) 满足下面的方程

$$\ddot{x}=2\dot{y}+x-\frac{\mu^*(x+\mu)}{r_1^3}-\frac{\mu(x-\mu^*)}{r_2^3},\quad \ddot{y}=-2\dot{x}+y-\frac{\mu^* y}{r_1^3}-\frac{\mu y}{r_2^3}$$

其中 $\mu=1/82.45$, $\mu^*=1-\mu$, $r_1=\sqrt{(x+\mu)^2+y^2}$, $r_1=\sqrt{(x-\mu^*)^2+y^2}$, 试在初值 $x(0)=1.2$, $\dot{x}(0)=0$, $y(0)=0$, $\dot{y}(0)=-1.04935751$ 下进行仿真，并绘制出 Apollo 位置的 (x,y) 轨迹。

16) 求解常微分方程的经典算法 — 4 阶 Runge-Kutta 方法如下：

$$x_{k+1} = x_k + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

其中

$$K_1 = f(x_k, t_k),\ K_2 = f\left(x_k + \frac{h}{2}K_1, t_k + \frac{h}{2}\right)$$

$$K_3 = f\left(x_k + \frac{h}{2}K_2, t_k + \frac{h}{2}\right),\ K_4 = f(x_k + K_3, t_k + h)$$

式中 h 为计算步长，试用 MATLAB 编写出定步长 4 阶 Runge-Kutta 算法程序，对前面 Apollo 系统进行重新计算，绘制出轨迹图，并和 ode45() 函数的结果进行比较。

参 考 文 献

[1] Dongarra J J, Bunsh J R, Molor C B. LINPACK user's guide. Philadelphia: Society of Industrial and Applied Mathematics (SIAM), 1979

[2] Forsythe G E, Malcolm M A, Moler C B. Computer methods for mathematical computations. Englewood Cliffs: Prentice-Hall, 1977

[3] Garbow B S, Boyle J M, Dongarra J J, Moler C B. Matrix eigensystem routines – EISPACK guide extension, Lecture notes in computer sciences. Vol. 51, New York: Springer-Verlag, 1977

[4] Gilbert J R, Moler C B, and Schreiber R. Sparse matrices in MATLAB: design and implementation. SIAM Journal on Matrix Analysis and Applications, 1992, 13: 33-356

[5] Gregory R T, Karney D L. A collection of matrices for testing computational algorithms. New York: Wiley, 1969

[6] Klema V C, Laub A J. The singular value decomposition: its computation and some applications. IEEE Trans. Automatic Control, 1980, AC-25(1): 164-176

[7] Moler C B, Stewar G W. An algorithm for generalized matrix eigenvalue problems. SIAM Journal of Numerical Analysis, 1973, 10: 241-256

[8] Moler C B, Van Loan C F. Nineteen dubious ways to compute the exponential of a matrix. SIAM Review, 1979, 20: 801-836

[9] Nelder J A, Mead R. A simplex method for function minimization. Computer Journal, 7: 308-313

[10] Shampine L F, Watts H A, Davenport S M. Solving non-stiff ordinary differential equations – the state of the art. SIAM Review, 1976, 18: 377-441

[11] Smith B T, Boyle J M, Dongarra J J *et al.* Matrix eigensystem routines – EISPACK guide, Lecture notes in computer sciences. Vol. 6 (Second edition). New York: Springer-Verlag, 1976

第 4 章　控制系统的数学模型及其转换方法

4.1　线性控制系统模型的基本描述方法

控制系统的数学模型在控制系统研究中是相当重要的，要对系统进行仿真处理，首先应该知道系统的数学模型，然后才可以对之进行模拟，如果已知了系统的模型，才可以在此基础上设计一个合适的控制器，使得原系统的响应达到预期的效果。

在线性系统理论中一般常用的数学模型形式不是很多，最常用的有状态方程模型 (又称为系统的内部模型) 、传递函数模型 (系统的外部模型) 以及零极点模型等，而这些模型之间又有着某种内在的等效关系。在一些场合下需要用到其中一种模型，而另一场合下可能又需要另外的模型，例如如果想获得系统的根轨迹图时往往需要已知系统的传递函数模型，而在二次型指标的最优设计时往往又需要知道系统的状态方程模型，所以了解由一种模型到另外模型类型的转换方法也是很必要的。在本章中将着重介绍系统模型之间的相互转换方法、标准型转换及 MATLAB 实现。此外，在原来系统模型未知时，如果想通过对系统响应实测的数据来得出系统模型，则涉及系统辨识的问题。在本章中还将分别讨论连续和离散时间系统的辨识方法，并以两类有代表性的方法为例介绍控制系统的模型降阶问题。

4.1.1　控制系统的传递函数描述

动态系统一般是以微分方程来描述的，而线性系统又是以线性常微分方程来描述的。例如假设系统的输入信号为 $u(t)$，且输出信号为 $y(t)$，则此系统相应的微分方程可以写成

$$\begin{aligned} a_1\frac{d^n y(t)}{dt^n}+a_2\frac{d^{n-1}y(t)}{dt^{n-1}} &+\cdots+a_n\frac{dy(t)}{dt}+a_{n+1}y(t) \\ &= b_1\frac{d^m u(t)}{dt^m}+b_2\frac{d^{m-1}u(t)}{dt^{m-1}}+\cdots+b_{m+1}u(t) \end{aligned} \tag{4.1.1}$$

对此系统的微分方程作 Laplace 变换，则单输入单输出 (single-input single-output, 简称 SISO) 连续系统的传递函数可以如下描述

$$G(s)=\frac{Y(s)}{U(s)}=\frac{b_1s^m+b_2s^{m-1}+\cdots+b_{m+1}}{a_1s^n+a_2s^{n-1}+\cdots+a_ns+a_{n+1}} \tag{4.1.2}$$

对线性时不变 (linear time invariant, 简记为 LTI) 系统来说，式中 a_i 与 b_i 均为常数，且 $a_1 \neq 0$。这种系统在 MATLAB 下可以方便地由其分子和分母系数所构成的两个向量唯一地确定出来

```
num=[b_1, b_2, …, b_{m+1}];     den=[a_1, a_2, …, a_n, a_{n+1}];
```

可见，在这样的表示方法下，分子和分母向量的内容分别是原传递函数的分子分母系数的降幂排列。

例 4.1　若给定系统的传递函数为

$$G(s) = \frac{12s^3 + 24s^2 + 12s + 20}{2s^4 + 4s^3 + 6s^2 + 2s + 2}$$

则可以将其用 MATLAB 语句表示出来：

```
>> num=[12 24 12 20];   den=[2 4 6 2 2];
```

一般情况下，往往需要对系统的传递函数模型系数作首一化处理，亦即将整个系统的分子和分母同时除以 a_1，这样原系统的模型就变为

$$G(s) = \frac{\hat{b}_1 s^m + \hat{b}_2 s^{m-1} + \cdots + \hat{b}_{m+1}}{s^n + \hat{a}_2 s^{n-1} + \cdots + \hat{a}_n s + \hat{a}_{n+1}} \tag{4.1.3}$$

式中 $\hat{b}_i = b_i/a_1$, $i = 1, \cdots, m+1$, 且 $\hat{a}_i = a_i/a_1$, $i = 2, \cdots, n+1$。这时，此系统的 MATLAB 表示为

```
num=[b̂_1, b̂_2, …, b̂_{m+1}];     den=[1, â_2, …, â_n, â_{n+1}];
```

例如对例 4.1 中给出的传递函数进行首一化处理之后则可以容易地表示成

```
>> num=[6 12 6 10]; den=[1 2 3 1 1]
```

当然，式 (4.1.3) 中给出的传递函数模型只是一种简单的特例，其它复杂的表示方式可以由下面的例子看出来

$$G_1(s) = \frac{4(s+2)(s^2+6s+6)^2}{s(s+1)^3(s^3+3s+2s+5)}$$

这时，整个系统的传递函数 $G_1(s)$ 就不再可以由上面叙述的系数降幂排列的形式简单地构造出来了，而必须借助 MATLAB 提供的多项式乘法运算函数 conv() 来处理以便获得分子和分母多项式向量。MATLAB 下的多项式乘法处理函数调用方式为：

```
c = conv(a, b)
```

其中 a 和 b 分别表示一个多项式，而 c 为 a 和 b 多项式的乘积多项式。例如给定两个多项式 $A(s) = s^3 + 2s^2 + 3s + 4$ 和 $B(s) = 10s^2 + 20s + 30$, 若想求出多项式 $C(s)$ 满足 $C(s) = A(s)B(s)$，则应该首先构造多项式 $A(s)$ 和 $B(s)$，然后再调用 conv() 函数来求出 $C(s)$



```
>>  A = [1, 2, 3, 4]; B = [10, 20, 30];
>>  C = conv(A, B)
C =    10     40    100    160    170    120
```

可以看出，最终得出的 $C(s)$ 多项式为 $10s^5+40s^4+100s^3+160s^2+170s+120$。MATLAB 提供的 conv() 函数的调用是允许多级嵌套的，例如前面的 $G_1(s)$ 可以由下面的语句来输入

```
>> num1=4*conv([1,2],conv([1,6,6],[1,6,6]));
>> den1=conv([1,0],conv([1,1],conv([1,1],conv([1,1],[1,3,2,5]))));
```

为简单起见，分母多项式还可以由下面的语句来输入

```
den1=[conv([1,1],conv([1,1],conv([1,1],[1,3,2,5]))) 0];
```

用户还可以进一步地编写一个 convs() 函数来一次性地求出若干个多项式的连乘积，该函数的清单如下

```
function a=convs(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10)
a=a1;
for i=2:nargin
   eval(['a=conv(a,a' int2str(i) ');']);
end
```

这样前面的传递函数就可以由下面的语句直接输入并得出如下结果

```
>>  num1=4*convs([1,2],[1,6,6],[1,6,6])
num1 =    4    56    288    672    720    288
>>  den1=convs([1,0], [1,1], [1,1],[1,1],[1,3,2,5])
den1 =    1     6     14     21     24     17     5     0
```

相应地，离散时间系统的动态模型一般是以差分方程来描述的，假设在第 i 采样时刻系统的输入信号为 $u(iT)$，且输出信号为 $y(iT)$，其中 T 为采样周期，则此系统相应的差分方程模型可以写成

$$\begin{aligned}g_1y[(i+n)T]+\quad & g_2y[(i+n-1)T]+\cdots+g_ny[(i+1)T]+g_{n+1}y(iT)\\ =\quad & f_1u[(i+m+1)T]+f_2u[(i+m)T]+\cdots+f_{m+1}u(iT)\end{aligned} \tag{4.1.4}$$

对上述差分方程模型进行 Z 变换，则可以得出系统的脉冲传递函数

$$G(z)=\frac{f_1z^m+f_2z^{m-1}+\cdots+f_{m+1}}{g_1z^n+g_2z^{n-1}+\cdots+g_nz+g_{n+1}} \tag{4.1.5}$$

对 LTI 系统来说，上式中 f_i 与 g_i 均为常数，且 $g_1\neq 0$。这种系统在 MATLAB 下也可以由其分子和分母系数构成的两个向量来唯一确定

```
num=[f1, f2, ..., fm+1];     den=[g1, g2, ..., gn, gn+1];
```

同样也可以对之进行首一化处理

$$G(z)=\frac{\hat{f}_1z^m+\hat{f}_2z^{m-1}+\cdots+\hat{f}_{m+1}}{z^n+\hat{g}_2z^{n-1}+\cdots+\hat{g}_nz+\hat{g}_{n+1}} \tag{4.1.6}$$

式中 $\hat{f}_i = f_i/g_1, i = 1, \cdots, m+1$, 且 $\hat{g}_i = g_i/g_1, i = 1, \cdots, n+1$ 。这时，此系统的 MATLAB 表示为

```
num=[f̂1, f̂2, ···, f̂m+1];      den=[1, ĝ2, ···, ĝn, ĝn+1];
```

在脉冲传递函数中，除了该传递函数的分子和分母系数之外，一般还常常涉及到采样周期的概念，在不会产生误解的情况下，假设 $g_1 = 1$, 则式 (4.1.4) 中的差分方程可以简写成

$$y_{i+n} + g_2 y_{i+n-1} + \cdots + g_n y_{i+1} + g_{n+1} y_i = f_1 u_{i+m+1} + f_2 u_{i+m} + \cdots + f_{m+1} u_i \tag{4.1.7}$$

对于多输入多输出 (multi-input multi-output, 简称 MIMO)LTI 系统来说，原系统的传递函数模型实际上是传递函数矩阵模型 [12]，传递函数矩阵可以有多种表示方式，其中最常用的是矩阵分式描述 (matrix fraction description, 简称 MFD) 方法，当然 MFD 有各种标准型，各种标准型有自己的应用范围，并且各种标准型之间是可以进行转换的 [5] 。英国剑桥大学 Jan Maciejowski 等学者为 MATLAB 设计了一个专门的多变量系统设计工具箱来处理这样的系统并进行辅助设计 [9], 一些基本内容可以参见第 6 章。

4.1.2 控制系统的状态方程模型

状态方程是描述控制系统的一种常用的方式，这种方式由于是基于系统的不可见的状态变量的，所以又往往称为系统的内部描述方法。连续 LTI 系统的状态方程可以写成

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t) \tag{4.1.8}$$

式中 $x(t) \in \mathbb{R}^{n\times 1}$ 称为状态向量，而 n 称为系统的阶次。$A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{q\times n}$, $D \in \mathbb{R}^{q\times p}$ 分别为常数矩阵，而系统的输入与输出个数分别为 p 和 q 。和前面介绍的单变量系统不同，这里给出的输入和输出的涵义更广，它们分别为 p 维和 q 维向量，而不是前面的标量信号。对单输入单输出系统来说，我们有 $p = q = 1$ 。在一般情况下，控制系统的状态方程还可以简记为 (A, B, C, D) ，如果 $D \equiv 0$ ，则系统的状态方程模型可以简记为 (A, B, C) 。

例 4.2 考虑一个双输入双输出系统的状态方程模型

$$\dot{x} = \begin{bmatrix} 2.25 & -5 & -1.25 & -0.5 \\ 2.25 & -4.25 & -1.25 & -0.25 \\ 0.25 & -0.5 & -1.25 & -1 \\ 1.25 & -1.75 & -0.25 & -0.75 \end{bmatrix} x + \begin{bmatrix} 4 & 6 \\ 2 & 4 \\ 2 & 2 \\ 0 & 2 \end{bmatrix} u, \quad y = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix}$$

这一系统的状态方程模型可以由下面的 MATLAB 命令很直观地表示出来

```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
     0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
>> B=[4, 6; 2, 4; 2, 2; 0, 2];
>> C=[0, 0, 0, 1; 0, 2, 0, 2];
>> D=zeros(2,2);
```




对于离散时间系统来说，状态方程模型可以写成

$$x[(i+1)T]=Fx(iT)+Gu(iT),\quad y[(i+1)T]=Cx[(i+1)T]+Du[(i+1)T] \tag{4.1.9}$$

在没有冲突的情况下，上面的状态方程还可以简单地写成

$$x_{i+1}=Fx_i+Gu_i,\quad y_{i+1}=Cx_{i+1}+Du_{i+1} \tag{4.1.10}$$

同样地，离散时间状态方程模型也可以简记为 (F,G,C,D) 或 (F,G,C)。

4.1.3 控制系统的零极点模型

零极点模型实际上是传递函数模型的另一种表现形式，其原理是分别对原系统传递函数的分子和分母进行分解因式处理，以获得系统的零极点表示形式。对单输入单输出系统来说，可以简单地将其零极点模型写作

$$G(s)=K\frac{\sum_{i=1}^{m}(s+z_i)}{\sum_{i=1}^{n}(s+p_i)}=K\frac{(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)} \tag{4.1.11}$$

式中 $z_i,i=1,2,\cdots,m$ 和 $p_i,i=1,2,\cdots,n$ 分别称为系统的零点和极点，它们既可以为实数，又可以为复数，而 K 称为系统的增益。系统的零极点模型可以被直接用来判断系统的稳定性,\如果系统的所有极点都位于左半 s- 平面，即 $\Re e\{p_i\}<0,i=1,2,\cdots,n$, 则称该系统是稳定的，否则则称系统是不稳定的。如果稳定系统所有的零点都位于左半 s- 平面，即 $\Re e\{z_i\}<0,i=1,2,\cdots,m$, 则称该系统为最小相位系统，否则称为非最小相位系统。如果系统的某个零点的值恰好等于其中一个极点的值，则它们之间可以对消以直接获得一个完全等效的低阶系统。

由式 (4.1.3) 给出的传递函数来说，分别对分子和分母多项式作因式分解，则可以得出系统的零极点模型。在 MATLAB 语言中提供了多项式求根的函数 `roots()`，其调用方法在前面已经介绍过了：`z=roots(p)`, 其中 p 为多项式的系数向量，而 z 为该多项式的各个根所构成的向量。系统的增益 K 即为原传递函数分子的最高项系数与分母最高项系数的比值。

对于式 (4.1.5) 所示的脉冲传递函数来说，也可以用类似的方法直接获得其零极点表示模型，这时若系统的全部极点都处于单位圆内，即 $|p_i|<1,i=1,2,\cdots,n$, 则称该系统为稳定系统，否则称系统为不稳定系统。同样若稳定系统的全部零点均位于单位圆内，则称系统为最小相位系统。

对多变量系统来说，其零极点模型的形式稍微复杂一些，在 MATLAB 控制系统工具箱下，只能直接处理单输入多输出 (SIMO) 的模型，假设在单一输入下系统传递函数矩阵的公分母为

$$D(s)=s^n+a_2s^{n-1}+\cdots+a_ns+a_{n+1} \tag{4.1.12}$$

则利用 MATLAB 提供的求根运算可以容易地获得其极点的表示形式，同样对各个分子多项式进行求根运算，则可以得出系统的零点表示形式。对于多输入多输出 (MIMO) 系统来说，应该分别对各个输入信号求出系统的零极点模型，最后才可以获得整个系统的零极点模型。

MATLAB 的控制系统工具箱提供了一个转换函数 ss2zp()[1]，其调用格式为

```
[Z, P, K]=ss2zp(A, B, C, D, iu)
```

用来由系统的状态方程模型求取系统的零极点模型，在此函数中， A，B，C，D 分别表示系统状态方程模型的各个矩阵，而 Z，P，K 为所得出系统的零点、极点和增益矩阵。当然原模型可以是 MIMO 形式，而 iu 表示要求的输入序号，若系统为单输入系统，则有 iu=1 。

例 4.3 重新考虑例 4.2 中给出的状态方程模型，分别对两个输入信号进行转换，则可以得出如下结果

```
>>  [z1,p1,k1]=ss2zp(a,b,c,d,1)
z1 = -1.5000     -1.0000 + 1.2247i
     -1.5000     -1.0000 - 1.2247i
      Inf        -1.5000
p1 = -0.5000 + 0.8660i
     -0.5000 - 0.8660i
     -1.5000 + 0.0000i
     -1.5000 - 0.0000i
k1 =  1.0000
      4.0000
>>  [z2,p2,k2]=ss2zp(a,b,c,d,2)
z2 =    -0.8750 + 0.6960i  -0.8333 + 0.8498i
        -0.8750 - 0.6960i  -0.8333 - 0.8498i
        -1.5000             -1.0000
p2 =    -0.5000 + 0.8660i
        -0.5000 - 0.8660i
        -1.5000 + 0.0000i
        -1.5000 - 0.0000i
k2 =     2.0000
        12.0000
```

可以看出在这个例子中，很巧合地有 d2=d1，记 $d(s)=(s+0.5\pm0.866i)(s+1.5)^2$，则整个系统的传递函数矩阵可以写成

$$G(s)=\frac{1}{d(s)}\begin{bmatrix}(s+1.5)^2 & 4(s+1.5)(s+1\pm1.2247i)\\ 2(s+1.5)(s+0.875\pm0.696i) & 12(s+0.8333\pm0.8498i)(s+1)\end{bmatrix}$$

[1]这里 “2” 应读为 “to”，意为到 ⋯ 转换，而 ss 为状态方程 (state space) 的缩写， zp 为零极点模型 (zero-pole) 的缩写，所以从该函数名可以容易地理解 ss2zp 的意义，即从状态方程到零极点模型的转换。MATLAB 的控制系统工具箱中还常用到另外一个缩写 tf，其含义为传递函数模型 (transfer function)，所以，这些函数名还是很便于记忆的。

系统的零极点模型实际上是一种中间模型，在MATLAB下经常将它转换到其它的形式下存储起来。控制系统工具箱中提供了一些将该模型转换到其它模型下的函数，如

```
[A, B, C, D]=zp2ss(Z, P, K);
```

其中Z，P，K分别给出系统的零极点和增益。该函数可以将给定零极点模型转换成状态方程模型 (A,B,C,D) 。控制系统工具箱还提供了零极点模型与传递函数模型之间的转换函数

```
[Z, P, K]=tf2zp(num, den),  及  [num, den]=zp2tf(Z, P, K)
```

其中前一个函数可以将传递函数num，den转换成零极点表示形式，而后一个函数可以将零极点表示方式转换成传递函数模型。

例4.4　考虑例4.1中给出的模型，调用前面给出的tf2zp()函数则可以直接获得系统的零极点模型

```
>> [z,p,k]=tf2zp(num,den)
z = -1.9294
    -0.0353 + 0.9287i
    -0.0353 - 0.9287i
p = -0.9567 + 1.2272i
    -0.9567 - 1.2272i
    -0.0433 + 0.6412i
    -0.0433 - 0.6412i
k =    6
```

亦即，变换后所得的零极点模型为

$$G(s)=6\frac{(s+1.9294)(s+0.0353\pm 0.9287i)}{(s+0.9567\pm 1.2272i)(s-0.0433\pm 0.6412i)}$$

为了验证这里MATLAB提供的转换函数，还可以调用zp2tf()函数将得出的模型变换回原来的传递函数模型。

```
>> [num1,den1]=zp2tf(z,p,k)
num1 =      0    6.0000   12.0000    6.0000   10.0000
den1 =1.0000    2.0000    3.0000    1.0000    1.0000
>> norm([0 num/2]-num1)
ans =  1.7585e-014
>> norm(den/2-den1)
ans =  1.6910e-015
```

可见，变换后的结果可以按相当高的精度又返回到原来的形式。

4.1.4 控制系统的典型连接

一般情况下已知的控制系统结构不是像前面一样简单，例如经常已知通过串联连接、并联连接及反馈连接的系统模型，为了对各种连接模式下系统能够进行方便的处



理，就需要获得该系统的单一模型。假设两个子系统模型分别为

$$\dot{x}_1 = A_1x_1+B_1u_1,\ y_1 = C_1x_1+D_1u_1,\ 和\ \dot{x}_2 = A_2x_2+B_2u_2,\ y_2 = C_2x_2+D_2u_2 \quad (4.1.13)$$

下面我们将讨论在各种连接模式下如何进行处理。

- **串联连接**: 在串联连接下显然有 $u_2 = y_1$, 且系统的输入和输出信号分别为 u_1 和 y_2, 这时将 $u_2 = y_1$ 代入第 2 子系统，则可见

$$\dot{x}_2 = A_2x_2 + B_2C_1x_1 + B_2D_1u_1 \quad (4.1.14)$$

  这时系统的整体模型为

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}A_1 & 0\\ B_2C_1 & A_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}B_1\\ B_2D_1\end{bmatrix}u_1,\ \ y_2 = [D_2C_1, C_2]\begin{bmatrix}x_1\\ x_2\end{bmatrix} + D_2D_1u_1 \quad (4.1.15)$$

- **并联连接**: 在并联连接下显然有 $u_1 = u_2$, $y = y_1 + y_2$, 且系统输入和输出信号分别为 u_1 和 y, 这样可以很容易地得出系统在并联连接下的整体状态方程模型

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}A_1 & 0\\ 0 & A_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}B_1\\ B_2\end{bmatrix}u_1,\ \ y = [C_1, C_2]\begin{bmatrix}x_1\\ x_2\end{bmatrix} + (D_1 + D_1)u_1 \quad (4.1.16)$$

- **负反馈连接**: 在负反馈连接下 $u_1 = r - y_2$, $u_2 = y_1$, 且系统的输入和输出信号分别为 r 和 y_1, 这时可以证明，系统的整体状态方程模型为

$$\begin{aligned}\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} &= \begin{bmatrix}A_1 - B_1ZD_2C_1 & -B_1ZC_2\\ B_2(I - D_1ZD_2)C_1 & A_2 - B_2D_1ZC_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}B_1Z\\ B_2D_1Z\end{bmatrix}r\\ y &= [(I - D_1ZD_2)C_1,\ -D_1ZC_2]\begin{bmatrix}x_1\\ x_2\end{bmatrix} + (D_1Z)r\end{aligned} \quad (4.1.17)$$

  式中 $Z = (I + D_1D_2)^{-1}$ 。若 $D_1 = D_2 = 0$, 则系统模型可以简化成

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}A_1 & -B_1ZC_2\\ B_2C_1 & A_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}B_1Z\\ 0\end{bmatrix}r\ \ y = [C_1,\ 0]\begin{bmatrix}x_1\\ x_2\end{bmatrix} \quad (4.1.18)$$

  当然对正反馈连接也有相应的结论。

值得指出的是由上面方法构造的整体状态方程并不一定是最小实现形式，亦即它们往往可以进一步化简。在 MATLAB 的控制系统工具箱中提供了系统的连接处理函数 series(), parallel() 和 feedback(), 它们的调用格式都是很直观的，以 series() 函数为例，其基本调用格式为

```
[A,B,C,D]=series(A1,B1,C1,D1,A2,B2,C2,D2)
```

其中 (A1,B1,C1,D1) 和 (A2,B2,C2,D2) 分别为第 1 和第 2 子系统状态方程模型，而 (A, B, C, D) 为系统的整体状态方程模型。若已知子系统的 SISO 传递函数模型，则可以由下面的格式来调用各个函数

```
[num,den]=series(num1,den1,num2,den2)
```

其中 (num1,den1) 和 (num2,den2) 分别为两个子系统的传递函数模型，而返回的 (num, den) 为系统整体的传递函数模型。

例 4.5 考虑下面的例子，假设两个子系统的状态方程模型分别为

$$\dot{x}_1=\begin{bmatrix}0&1\\1&-2\end{bmatrix}x_1+\begin{bmatrix}0\\1\end{bmatrix}u_1,\ y_1=[1,3]x_1+u_1,\ \ \dot{x}_2=\begin{bmatrix}0&1\\-1&-3\end{bmatrix}x_2+\begin{bmatrix}0\\1\end{bmatrix}u_2,\ y_2=[1,4]x_2+3u_2$$

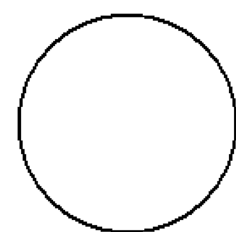
则通过下面的函数调用可以得出各种连接下系统的整体状态方程模型

```
>> a1=[0 1;-1 -2]; b1=[0;1]; c1=[1,3]; d1=1;
>> a2=[0 1;-1 -3]; b2=b1; c2=[1 4]; d2=3;
>> [a,b,c,d]=series(a1,b1,c1,d1,a2,b2,c2,d2)
a =   0     1     0     0
     -1    -2     0     0
      0     0     0     1
      1     3    -1    -3
b =   0
      1
      0
      1
c =   3     9     1     4
d =   3
>> [a,b,c,d]=parallel(a1,b1,c1,d1,a2,b2,c2,d2)
a =   0     1     0     0
     -1    -2     0     0
      0     0     0     1
      0     0    -1    -3
b =   0
      1
      0
      1
c =   1     3     1     4
d =   4
>> [a,b,c,d]=feedback(a1,b1,c1,d1,a2,b2,c2,d2)
a =          0     1.0000          0          0
       -1.7500    -4.2500    -0.2500    -1.0000
             0          0          0     1.0000
        0.2500     0.7500    -1.2500    -4.0000
b =          0
        0.2500
             0
        0.2500
c =     0.2500     0.7500    -0.2500    -1.0000
d =     0.2500
```




若两个子系统为正反馈连接，则

```
>> [a,b,c,d]=feedback(a1,b1,c1,d1,a2,b2,c2,d2,+1)
a =        0    1.0000         0         0
     -2.5000   -6.5000   -0.5000   -2.0000
           0         0         0    1.0000
     -0.5000   -1.5000   -1.5000   -5.0000
b =        0
     -0.5000
           0
     -0.5000
c =  -0.5000   -1.5000   -0.5000   -2.0000
d =  -0.5000
```



4.1.5 控制系统的结构图描述及转换

前面介绍的几种模型表述方法都是对整个模型的集中表示，若一个系统模型由若干个相互关联的子系统所构成，比如整个系统是由如图 4-1 所示的结构图形式构成，则为了对之进行整体地处理，首先需要将该系统转换成传递函数或状态方程这样的集中表述。

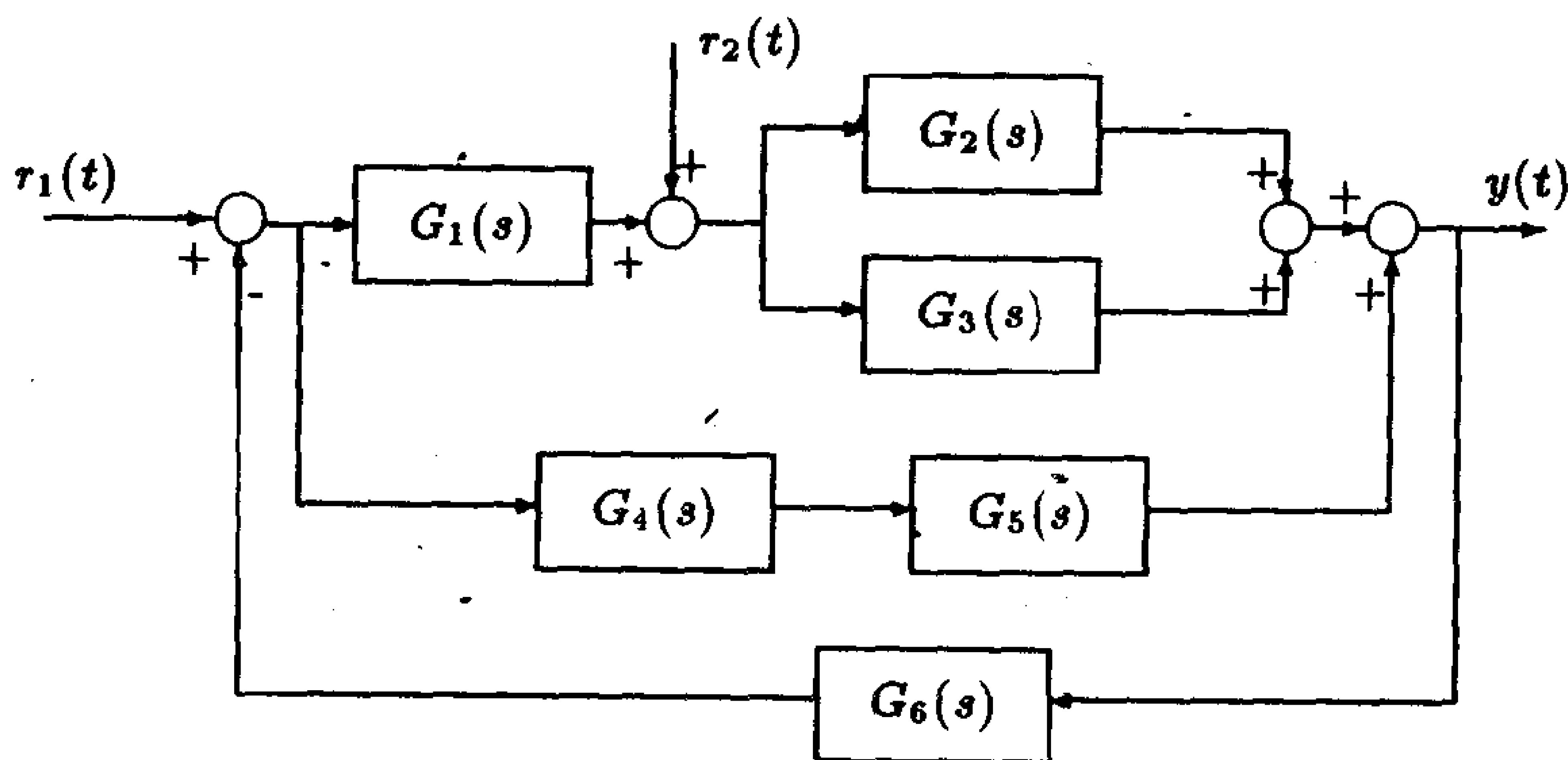


图 4-1　控制系统的框图表示方式

假设系统共由 M 个通路构成，且已知所有通路的子状态方程，可假定第 i 个通路为

$$\dot{x}_i = A_i x_i + B_i u_i,\quad y_i = C_i x_i + D_i u_i \tag{4.1.19}$$

这样就可以按下列方式扩展出新的矩阵 $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$，满足

$$\tilde{A} = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_M \end{bmatrix},\ \tilde{B} = \begin{bmatrix} B_1 & & \\ & \ddots & \\ & & B_M \end{bmatrix},\ \tilde{C} = \begin{bmatrix} C_1 & & \\ & \ddots & \\ & & C_M \end{bmatrix},\ \tilde{D} = \begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_M \end{bmatrix} \tag{4.1.20}$$

这样的工作可以由 MATLAB 控制系统工具箱中提供的 append() 函数来完成，该函数的调用格式为



```
[A, B, C, D] = append(A1, B1, C1, D1, A2, B2, C2, D2)
```

这时该函数的输入与输出矩阵之间的关系为

$$A=\begin{bmatrix}A1 & 0\\ 0 & A2\end{bmatrix},\quad B=\begin{bmatrix}B1 & 0\\ 0 & B2\end{bmatrix},\quad C=\begin{bmatrix}C1 & 0\\ 0 & C2\end{bmatrix},\quad D=\begin{bmatrix}D1 & 0\\ 0 & D2\end{bmatrix} \qquad (4.1.21)$$

系统各个环节之间的连接关系可以由下列方式来描述，首先列写出系统的关联矩阵 K，使得其第 (i,j) 元素代表第 i 通路与第 j 通路的关联情况，即若第 i 通路为第 j 通路输入的一个组成部分时，则 $K_{ij}=1$，否则其值为 0 。例如，图 4-1 中系统的关联矩阵可以写成

$$K=\begin{bmatrix}0&0&0&0&0&1\\1&0&0&0&0&0\\1&0&0&0&0&0\\0&0&0&0&0&1\\0&0&0&1&0&0\\0&1&1&0&1&0\end{bmatrix}$$

得出 K 矩阵后，可以容易地得出下列各个矩阵:

$$A=\widetilde{A}+\widetilde{B}K\widetilde{C},\quad \widehat{B}=\widetilde{B}(I-K\widetilde{D})^{-1},\quad \widehat{C}=(I-\widetilde{D}K)^{-1}\widetilde{C},\quad \widehat{D}=(I-\widetilde{D}K)^{-1}\widetilde{D} \qquad (4.1.22)$$

这时系统总体的状态方程 (A,B,C,D) 可以如下获得： B 矩阵取 $\widehat{B}$ 中含有输入通路的各列向量的和， C 矩阵为 $\widehat{C}$ 中含有输出通路的各行向量的和， D 矩阵为 $\widehat{D}$ 矩阵中有输入和输出通路关联的子矩阵。

例 4.6 考虑图 4-1 中所示的控制系统方框图，假设其中各个子传递函数的表达式为

$$G_1(s)=\frac{s+3}{s^2+2s+1},\ G_2(s)=\frac{1}{4s+2},\ G_3(s)=\frac{s+5}{s^2+3s+5},\ G_4(s)=\frac{1}{s},\ G_5(s)=5,\ G_6(s)=\frac{1}{5s+1}$$

则逐次调用 MATLAB 提供的 append() 函数，可以立即得出

```
AT = -2.0000   -1.0000         0         0         0         0         0
      1.0000         0         0         0         0         0         0
           0         0   -0.5000         0         0         0         0
           0         0         0   -3.0000   -5.0000         0         0
           0         0         0    1.0000         0         0         0
           0         0         0         0         0         0         0
           0         0         0         0         0         0   -0.2000
BT =     1     0     0     0     0     0
         0     0     0     0     0     0
         0     1     0     0     0     0
         0     0     1     0     0     0
         0     0     0     0     0     0
         0     0     0     1     0     0
         0     0     0     0     0     0
CT =  1.0000    3.0000         0         0         0         0         0
```




```
         0         0    0.2500         0         0         0         0
         0         0         0    1.0000    5.0000         0         0
         0         0         0         0         0    1.0000         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0   -0.2000
DT =     0    0    0    0    0    0
         0    0    0    0    0    0
         0    0    0    0    0    0
         0    0    0    0    0    0
         0    0    0    0    5    0
         0    0    0    0    0    0
```

其实从给出的 AT，BT，CT，DT 矩阵也可以明显地看出 append() 函数的作用。从上面的结果可以看出，得出的各个矩阵中非零元素只是其中的一小部分，所以若将 append() 函数作一下修改，在该函数中使用稀疏矩阵来取代现有的各个矩阵，则可以使存储空间大大地减小。输入了系统的关联矩阵之后，就可以根据式 (4.1.22) 容易地求出系统的状态方程模型

```
>>  K=[1,0,0,0,0,1; 1,0,0,0,0,0; 1,0,0,0,0,0; 0,0,0,0,0,1;
       0,0,0,1,0,0; 0,1,1,0,1,0];
>>  A=AT+BT*K*CT
A =  -2.0000   -1.0000         0         0         0         0   -0.2000
      1.0000         0         0         0         0         0         0
      1.0000    3.0000   -0.5000         0         0         0         0
      1.0000    3.0000         0   -3.0000   -5.0000         0         0
           0         0         0    1.0000         0         0         0
           0         0         0         0         0         0   -0.2000
           0         0    0.2500    1.0000    5.0000    5.0000   -0.2000
>>  BB = BT*inv(eye(6)-K*DT); ii1=[1,4]; ii2=[2,3];
>>  B=[sum(BB(:,ii1)')', sum(BB(:,ii2)')']
B =  1    0
     0    0
     0    1
     0    1
     0    0
     1    0
     0    0
>>  CC=inv(eye(6)-DT*K)*CT; ii=[2,3,5];
>>  C=sum(CC(ii,:))
C =   0         0    0.2500    1.0000    5.0000    5.0000         0
>>  DD=inv(eye(6)-DT*K)*DT;
>>  D=[sum(sum(DD(ii,ii1))), sum(sum(DD(ii,ii2)))]
D =   0    0
```

其中 ii1 表示和第 1 输入信号进入的所有各个通路的序号，而 ii2 表示输入 2 进入的所有通路序号。变量 ii 则表示输出信号是由哪些通路构成的。

从上面介绍的方法可以看出，其中列写系统的关联矩阵是一项很麻烦的工作，文献



[15] 叙述了作者编写的名为 BLOCKM 的结构图输入软件，该软件完全由 MATLAB 语句编写。在 BLOCKM 中实现了一种由用输入的结构图自动生成关联矩阵的功能，使得系统状态方程的计算更加可靠。此外，由 BLOCKM 还可以容易地计算出具有复杂结构的系统状态方程模型来。

例 4.7. 下面分析一个著名的 CSCAD 软件测试基准问题 (benchmark problem)：F-14 战斗机模型处理问题，其中 F-14 战斗机的结构图如图 4-2 所示，该系统中的参数如下

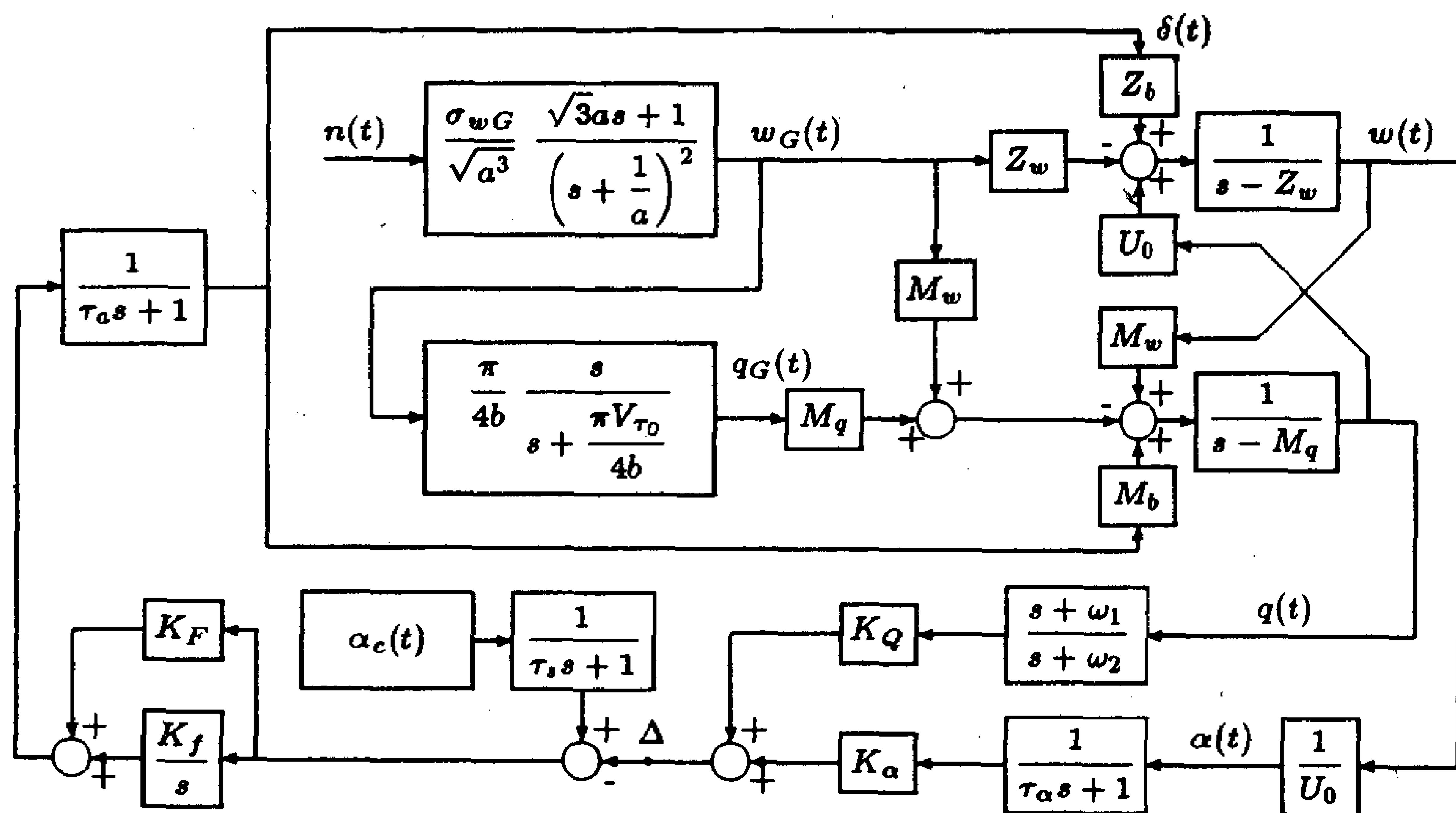


图4-2　F-14 战斗机系统数学模型

$$
\begin{array}{llll}
\tau_a = 0.05, & \sigma_{wG} = 3.0, & a = 2.5348, & b = 64.13 \\
V_{\tau_0} = 690.4, & \sigma_\alpha = 5.236 \times 10^{-3}, & Z_b = -63.9979, & M_b = -6.8847 \\
U_0 = 689.4, & Z_w = -0.6385, & M_q = -0.6571, & M_w = -5.92 \times 10^{-3} \\
\omega_1 = 2.971, & \omega_2 = 4.144, & \tau_s = 0.10, & \tau_\alpha = 0.3959 \\
K_Q = 0.8156, & K_\alpha = 0.6770, & K_f = -3.864, & K_F = -1.745
\end{array}
$$

将输入向量选为 $u = [n(t), \alpha_c(t)]^T$，其中 $n(t)$ 为单位均值的白噪声信号，而 $\alpha_c(t) = K\beta(\mathrm{e}^{-\gamma t} - \mathrm{e}^{-\beta t})/(\beta - \gamma)$ 为攻击角度命令输入信号，这里 $K = \alpha_{c_{\max}} \mathrm{e}^{\gamma t_m}$，且 $\alpha_{c_{\max}} = 0.0349$，$t_m = 0.025$，$\beta = 426.4352$，$\gamma = 0.01$ 。系统的输出向量为 $y(t) = [N_{Z_p}(t), \alpha(t), q(t)]^T$ ，这里 $N_{Z_p}(t)$ 信号定义为 $N_{Z_p}(t) = [-\dot{w}(t) + U_0 q(t) + 22.8\dot{q}(t)]/32.2$，则可以得出系统的状态方程模型

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t)$$

其中 [15]

$$
A = \begin{bmatrix}
-10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & -0.8156 & 0.9567 & -1.7100 & 0 & 0 & 0 \\
-17.45 & -3.864 & -20 & 0 & 1.4232 & -1.6695 & 2.9840 & 0 & 0 & 0 \\
0 & 0 & -1280 & -0.6385 & 689.4 & 0 & 0 & 2.0839 & 0.4746 & 0 \\
0 & 0 & -137.69 & -0.00592 & -0.6571 & 0 & 0 & 0.0456 & 0.01038 & -0.06804 \\
0 & 0 & 0 & 0 & 1 & -4.144 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.00145 & 0 & 0 & -2.5259 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.7890 & -0.1556 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.2637 & 0.7434 & -8.4553
\end{bmatrix}
$$

$$B=\begin{bmatrix}1&0\\0&0\\0&0\\0&0\\0&0\\0&0\\0&1\\0&0\\0&0\end{bmatrix},\quad C^T=\begin{bmatrix}0&0&0\\0&0&0\\-57.7474&0&0\\0.0156&0.0015&0\\-0.4653&0&1\\0&0&0\\0&0&0\\-0.0324&0&0\\-0.0074&0&0\\-0.0482&0&0\end{bmatrix}$$

由得出的状态方程模型可以容易地求出从输入 $u_c(t)$ 到各个输出信号的零极点表示

```
>>  [z,p,k]=ss2zp(A,B,C,D,2)
z = -13.0090 +14.2264i    -2.5259              -4.1440
    -13.0090 -14.2264i   -19.2348 +10.2853i -19.9645
     -7.6504             -19.2348 -10.2853i  -3.1349 + 0.6530i
     -0.2278              -5.7353              -3.1349 - 0.6530i
     -1.1155 + 1.4762i    -2.8527               1.3922
     -1.1155 - 1.4762i    -1.2797              -1.6355
     -3.0790              -0.2278             -10.0000
     -1.9393             -10.0000              -0.2278
    -10.0000                  Inf                  Inf
 p = -9.8425 + 9.5653i
     -9.8425 - 9.5653i
     -8.4553
     -1.6721 + 1.5969i
     -1.6721 - 1.5969i
     -1.8140
     -3.1224
     -0.3945 + 0.0000i
     -0.3945 - 0.0000i
    -10.0000
 k = -0.0324
      0.0030
      0.0456
```

对于含有非线性环节的原系统框图来说，还可以采用 SIMULINK 来完成这一工作，这需要首先绘制出系统的框图结构，然后再调用 SIMULINK 提供的线性化函数 linmod() 或 linmod2() 来实现。从 SIMULINK 提供的函数可以直接求出在选定的工作点处系统的线性化状态方程模型。关于 SIMULINK 的使用方法将在第 5 章内详细叙述。SIMULINK 中的演示程序中给出了一个 F-14 系统的结构图模型，如图 4-3 所示，其中一些模块下面还有下一级的子模块，在这里就不再叙述了。和原始系统相比，可以看出在这一模型中 $N(t)$ 输入并不是以输入端子的形式给出的，所以在本结构图中只有一个输入信号。利用线性化函数 linmod2() 并调用 ss2zp() 函数可以求出系统的零极点模型

```
>>  [A, B, C, D] = linmod2('f14');
>>  [z, p, k] = ss2zp(a, b, c, d, 1)
z = -74.8207             -8.4553
```

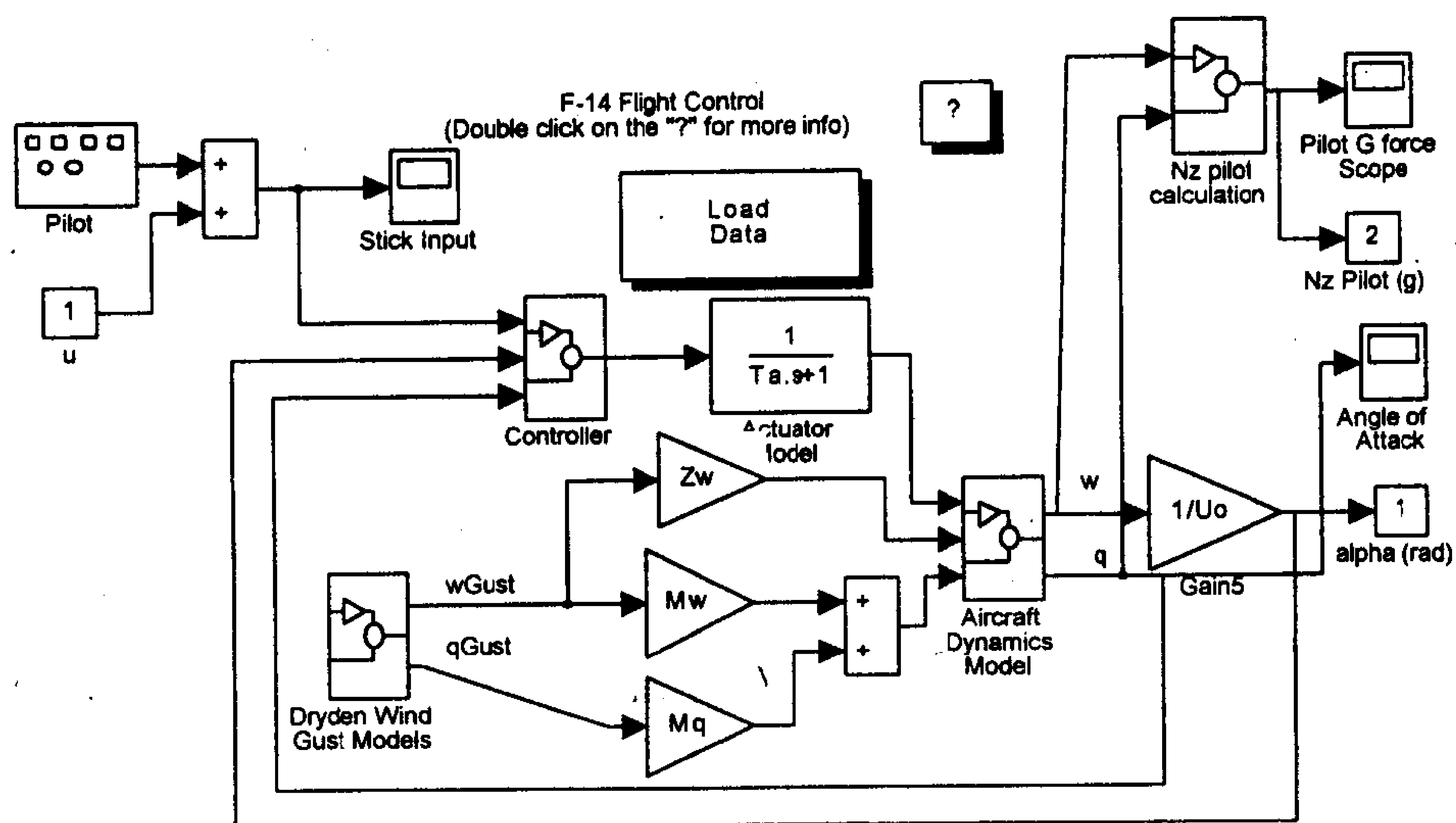
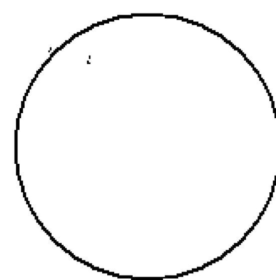


图4-3　F-14 系统的 SIMULINK 描述

```
       -4.1440            -4.1440
       -8.4553            -2.5259
       -2.5259            -0.5835
       -0.3945 + 0.0000i  -0.3945 + 0.0000i
       -0.3945 - 0.0000i  -0.3945 - 0.0000i
       -2.2131            -2.2131
p =    -9.8432 + 9.5718i
       -9.8432 - 9.5718i
       -1.6717 + 1.5970i
       -1.6717 - 1.5970i
       -3.1231
       -1.8126
       -0.3945 + 0.0000i
       -0.3945 - 0.0000i
       -8.4553
      -10.0000
k =      1.0e+004 *
         0.0032
         5.1472
```



可以看出，用线性化方法求出的零极点模型与 BLOCKM 求出的零极点是极其相近的，和 Rimer 等给出的精确解 [11] 相比较， BLOCKM 的结果更精确，因为它并未经过线性化这样的近似过程。

4.2 控制系统的稳定性分析

前面的分析中曾经介绍过，对线性系统来说，如果一个连续系统的所有极点都位于左半 s- 平面，则该系统是稳定的。

对离散系统来说，如果一个系统的全部极点都位于一个单位圆内，则此系统可以被认为是稳定的。

以前在多项式方程求根不是很容易时，往往采用间接的方法来判断系统的稳定性，例如构造 Routh 表或 Jury 表，这样可以由比较简单的方法间接地判断系统的稳定性。随着 MATLAB 这样具有高度数学计算功能的语言的出现，人们如果想判断线性系统的稳定性，一般就不再采用这样间接的方法了，因为可以提供一个 MATLAB 函数的调用直接求出控制系统的所有极点，而没有必要再去编写程序来间接地解决这样的问题了。

直接求根的判定方法除了能轻而易举地直接判定线性定常系统的稳定性之外，还可以同时判定所研究的系统是否为最小相位系统，所谓最小相位系统，对连续系统来说，就是系统本身是稳定的，且该系统的全部零点都位于左半 s- 平面，而对离散系统来说，即稳定系统的全部零点都位于一个单位圆内。很显然，最小相位的判定最直接的方法是根据系统的零点情况来进行。而系统的零点也同时可以由一个 MATLAB 函数的调用从原来系统的模型直接求出来，这样通过极其简单的判断就可以断定系统是否为最小相位系统。

例 4.8 考虑例 4.2 中给出的 2×2 系统，输入了系统状态方程的 A, B, C, D 之后，则可以由下面的语句来判定系统的稳定性

```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
    0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
>> B=[4, 6; 2, 4; 2, 2; 0, 2];
>> C=[0, 0, 0, 1; 0, 2, 0, 2]; D=zeros(2,2);
>> [Z1, P1, K1]=ss2zp(A,B,C,D,1);
>> [Z2, P2, K2]=ss2zp(A,B,C,D,2);
>> ii=find(real(P1)>0); n1=length(ii);
>> ii=find(real(P2)>0); n2=length(ii);
>> if (n1+n2>0), disp('System is Unstable');
   else, disp('System is Stable'); end
```

其实系统零极点在例 4.3 中已经求出来了，通过后面几个语句进行判断，就可以指出系统是否稳定。在上面的语句段中调用了 find() 函数，该函数是 MATLAB 中比较重要的一个函数，它用来得出满足指定条件的数组下标向量，该函数的调用格式为

```
ii = find(条件式)
```

例如这里的条件式为 real(P)>0，其含义为找出 P 数组 (矩阵) 中满足实部的值大于 0 的所有元素下标，并将结果返回到 ii 数组中去。这样如果找到了实部大于 0 的极点，则会将该极点的序号返回到 ii 下。如果最终的结果里 ii 的元素个数大于 0，则认为找到了不稳定极点，因而给出系统不稳定的提示，若产生的 ii 向量的元素个数为 0，则认为没有找到不稳定极点，因而得出系统稳定的结论。



对照例 4.3 中给出的结果不难看出，原系统是稳定的，因此将得到 System is Stable 字样的提示。采用直接方法对同样的系统还可以判定系统是否为最小相位系统，这样的判定可以通过下面的程序段来完成

```
>> ii=find(real(Z1)>0); n1=length(ii);
>> ii=find(real(Z2)>0); n2=length(ii);
>> if (n1+n2>0), disp('System is a Nonminimal Phase One');
   else, disp('System is a Minimal Phase One'); end
```

当然这样的判定语句和判定稳定性时使用的是一致的，所不同的是这里条件式中用到的是零点 Z1，Z2，而不是极点 P1，P2。同样对照例 4.3 中给出的结果可以看出，原系统是最小相位系统，因此将得到 System is a Minimal Phase One 字样的提示。

例 4.9 如果给出的系统模型为

$$G(s)=\frac{3s^4+2s^3+s^2+4s+2}{3s^5+5s^4+s^3+2s^2+2s+1}$$

当然可以采用下面的 MATLAB 语句来判定系统的稳定性

```
>> num=[3,2,1,4,2];  den=[3,5,1,2,2,1];
>> [z,p]=tf2zp(num,den)
z = 0.4500 + 0.9870i
    0.4500 - 0.9870i
   -1.0000
   -0.5666
p =-1.6067
    0.4103 + 0.6801i
    0.4103 - 0.6801i
   -0.4403 + 0.3673i
   -0.4403 - 0.3673i
>> ii=find(real(p)>0); n1=length(ii)
n1 =   2
>> if (n1>0), disp(['System is unstable, with ' int2str(n1) 'unstable poles']);
   else, disp('System is stable'); end
System is unstable, with 2 unstable poles
```

这里首先调用了 tf2zp() 函数来求出系统的零极点，然后用和前面同样的方法来判定不稳定极点的个数，可以得出不稳定极点个数为 n1=2，因而得出系统不稳定的提示，并显示不稳定极点个数。进一步地，还可以由下面的 MATLAB 语句指出系统不稳定极点所在的位置

```
>> disp('The Unstable poles are:'), disp(p(ii))
The Unstable poles are:
   0.4103 + 0.6801i
   0.4103 - 0.6801i
```

例 4.10 下面再分析给出的离散时间系统的稳定性，已知开环系统的模型为

$$G(z)=\frac{5z^2+4z^4+z^3+0.6z^2+3z+0.5}{z^5}$$

则可以通过下面的 MATLAB 语句求出系统的闭环传递函数模型

```
>>  numo=[5,4,1,0.6,3,0.5]; deno=[1,0,0,0,0,0];
>>  numc=numo; denc=deno+numo
denc = 6.0000    4.0000    1.0000    0.6000    3.0000    0.5000
```

这样就可以容易地对系统进行稳定性和最小相位性分析了

```
>>  [z, p]=tf2zp(numc,denc)
z = -0.7822 + 0.5660i
    -0.7822 - 0.5660i
     0.4681 + 0.6367i
     0.4681 - 0.6367i
    -0.1718
p = -0.7133 + 0.5552i
    -0.7133 - 0.5552i
     0.4659 + 0.6139i
     0.4659 - 0.6139i
    -0.1717
>>  ii=find(abs(p)>1); n1=length(ii);
>>  if (n1>0), disp('System is Unstable');
    else, disp('System is Stable'); end
System is Stable
>>  ii=find(abs(z)>1); n2=length(ii);
>>  if (n2>0), disp('System is a Nonminimal Phase One');
    else, disp('System is a Minimal Phase One'); end
System is a Minimal Phase One
```

可以看出，通过直接的判定方法可知，原系统为稳定的，同时也是最小相位系统。其实通过观察得出的零极点也可以直接获得这样的结论。

虽然在一些场合下，由线性系统求根方法来判定系统的稳定性是最简单不过的事情了，但确实有很多条件下求解 Lyapunov 方程的方式也是很必要的，尤其在系统含有非线性环节时更是如此。下面首先介绍 Lyapunov 稳定判据，然后叙述 Lyapunov 方程的求解问题。

对 LTI 系统状态方程模型 (A, B, C) 来说， Lyapunov 稳定性分析基于下面的假设：如果对任意给定的对称正定矩阵 W，均存在唯一的正定矩阵解 V 满足下面的方程

$$A^T V + VA = -W \tag{4.2.1}$$

上面的方程称为 Lyapunov 方程，求解 Lyapunov 方程的解法是多种多样的，在第 3 章中曾介绍过利用 Kronecker 乘积来求解相应的 Lyapunov 方程的算法， MATLAB 提供了 Lyapunov 方程的求解函数 lyap()，该函数是采用 Schur 方法编写的，其调用格式为

```
X=lyap(A, B, C);
```

其中更一般地此函数可以求解下面给出的 Lyapunov 方程

$$AX + XB = -C \tag{4.2.2}$$



若在调用 lyap() 函数时只给出两个参数 V=lyap(A,W)，则将获得式 (4.2.1) 给出的 Lyapunov 方程的解。

4.3 状态方程与传递函数的相互转换

如果给定了 LTI 状态方程模型 (A,B,C,D)，则可以容易地表示系统的传递函数模型

$$G(s)=C\left[sI-A\right]^{-1}B+D \tag{4.3.1}$$

其中 I 为 n 阶单位矩阵。由式 (4.3.1) 可见，求取传递函数的关键在于求解 $\left[sI-A\right]^{-1}$。其中的分母多项式系数可以由第 3 章中介绍的 Fadeev-Fadeeva 算法直接求出，此外传递函数的分子多项式系数可以表示成 $N_i=CR_iB$, $i=1,2,\cdots,n$, 这样就可以容易地求出系统的传递函数为

$$G(s)=\frac{N_1s^{n-1}+N_2s^{n-2}+\cdots+N_{n-1}s+N_n}{s^n+a_1s^{n-1}+a_2s^{n-2}+\cdots+a_{n-1}s+a_n} \tag{4.3.2}$$

MATLAB 的控制系统工具箱中提供了大量的实用函数，其中 ss2tf() 函数可以方便地由给定的状态方程模型求出传递函数模型。这一函数的调用方式为

```
[num, den]=ss2tf(A, B, C, D, iu);
```

其中 A, B, C, D 矩阵表示系统的状态方程模型，而 iu 为输入的代号，对于单输入系统来说， iu=1 。对于多变量系统来说，不能用此函数一次地求出对所有输入信号的整个传递函数矩阵，而必须对各个输入信号逐个地求取传递函数子矩阵，最后获得整个的传递函数矩阵。

例 4.11 对于例 4.2 中给出的多变量系统来说，可以由下面的命令分别对各个输入信号求取传递函数向量，然后求出这个传递函数矩阵

```
>>  [num1, den1] = ss2tf(A, B, C, D, 1)
num1 = 0        0.0000     1.0000     3.0000     2.2500
       0        4.0000    14.0000    22.0000    15.0000
den1 = 1.0000   4.0000     6.2500     5.2500     2.2500

>>  [num2, den2] = ss2tf(A, B, C, D, 2)
num2 =  0         2.0000     6.5000     7.7500     3.7500
        0        12.0000    32.0000    37.0000    17.0000
den2 = 1.0000     4.0000     6.2500     5.2500     2.2500
```

对这个例子来说，恰好有 den2=den1，这样原系统对应的传递函数矩阵可以写成

$$G(s)=\frac{1}{s^4+4s^3+6.25s^2+5.25s+2.25}\begin{bmatrix} s^2+3s+2.25 & 4s^3+14s^2+22s+15 \\ 2s^3+6.5s^2+7.75s+3.75 & 12s^3+32s^2+37s+17 \end{bmatrix}$$



例 4.12　若给定一个离散时间状态方程

$$\dot{x}_{k+1}=\begin{bmatrix}1 & 2 & -1\\ 0 & 1 & 0\\ 1 & -4 & 3\end{bmatrix}x_k+\begin{bmatrix}0\\0\\1\end{bmatrix}u_k,\quad y_k=[1\quad -1\quad 1]x_k$$

则可以容易地由 ss2tf() 函数求出其脉冲传递函数

```
>>  [num,den]=ss2tf(f,g,c,d,1)
num =     0     1     -3     2
den =     1    -5      8    -4
```

如果已知系统的传递函数模型，求取系统状态方程模型的过程又称为系统的实现。由于状态变量可以较任意地选择，所以实现的方法并不是唯一的，这里只介绍比较常用的一种方法。我们知道，式 (4.1.2) 中给出的传递函数模型对应于式 (4.1.1) 中的微分方程模型，适当地选择系统的状态变量，且假设系统已经作了首一化处理即 $a_1=1$, 则系统的状态方程模型可以直接写成

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \vdots\\ \dot{x}_n\end{bmatrix}=\begin{bmatrix}-a_2 & -a_3 & \cdots & -a_{n+1}\\ 1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ \vdots\\ x_n\end{bmatrix}+\begin{bmatrix}1\\0\\0\\ \vdots\\0\end{bmatrix}u,\quad y=\Big[\overbrace{0,\cdots}^{\text{n-m-1 个 0}},b_1,\cdots,b_m\Big]\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix} \tag{4.3.3}$$

这种方法又称为可控标准型实现方法，在 MATLAB 的控制系统工具箱中使用的就是这种方法。可控标准型实现的框图表示如图 4-4 所示。这样的转换可以通过调用函数

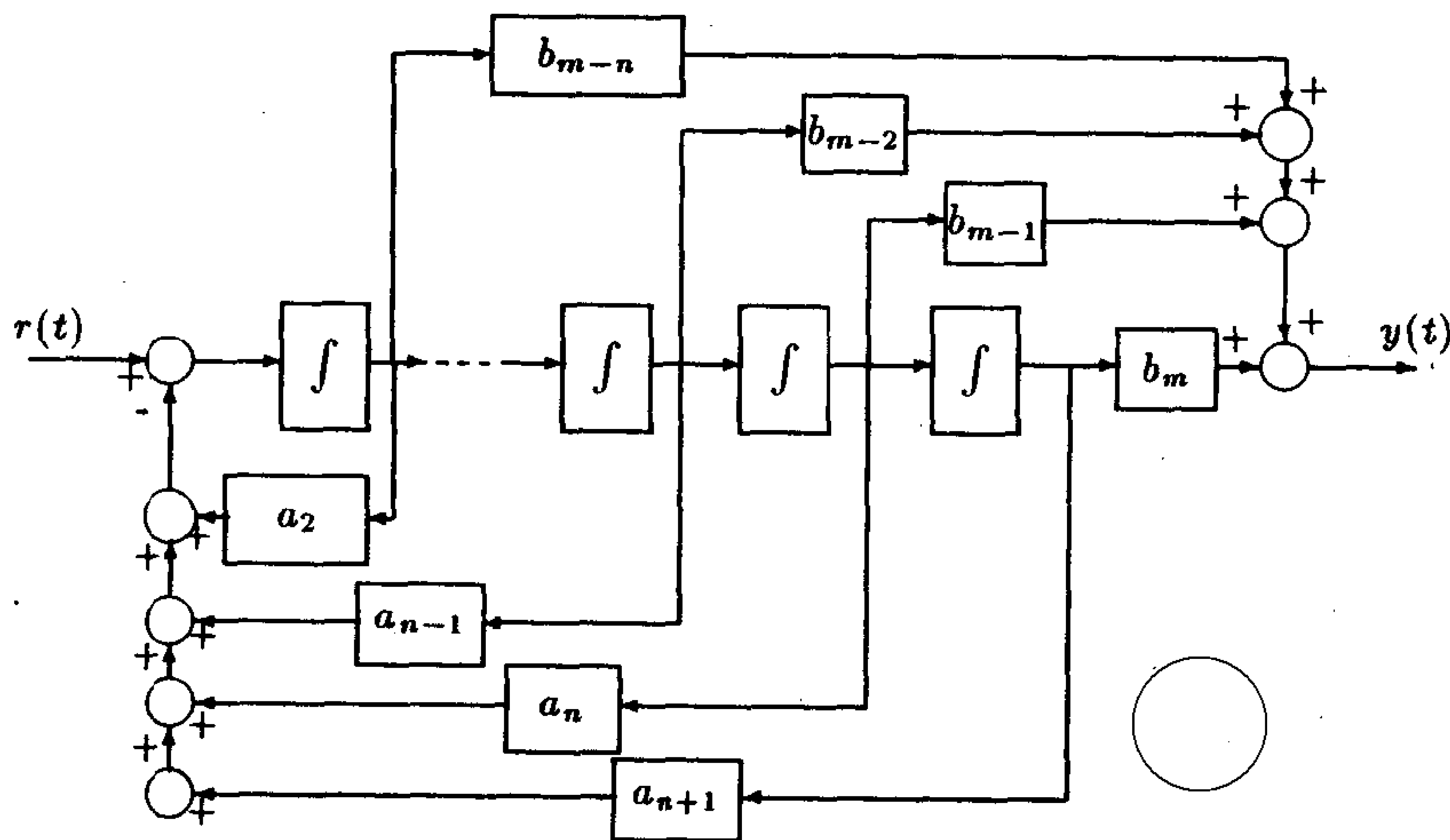


图4-4　可控标准型的框图表示

tf2ss() 来实现

```
[A, B, C, D] = tf2ss(num, den)
```

可以直观地看出，系统的传递函数模型由 num, den 所给出的分子和分母多项式来定义，调用此函数后会自动返回系统的状态方程模型 (A, B, C, D)。

例 4.13　如果系统的传递函数模型为

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

则通过下面的语句将可以得出系统的状态方程模型

```
>>  num = [1,7,24,24]; den = [1,10,35,50,24];
>>  [A, B, C, D] = tf2ss(num, den)
A =   -10    -35    -50    -24
        1      0      0      0
        0      1      0      0
        0      0      1      0
B =     1
        0
        0
        0
C =     1      7     24     24
D =     0
```

4.4　控制系统模型的连续化与离散化

假设连续 LTI 系统的状态方程为

$$\dot{x}(t)=Ax(t)+Bu(t),\quad y(t)=Cx(t)+Du(t) \tag{4.4.1}$$

且离散时间系统的状态方程为

$$x_{k+1}=Fx_k+Gu(t),\quad y_k=\widehat{C}x_k+\widehat{D}u_k \tag{4.4.2}$$

并保证 $y(t)=y_k$，其中 $t=kT$, 则有 $\widehat{C}=C,\ \widehat{D}=D$ 。这时可以容易地证明

$$F=\mathrm{e}^{AT},\quad G=\int_0^T \mathrm{e}^{A(T-\tau)}B\mathrm{d}\tau \tag{4.4.3}$$

由 MATLAB 的控制系统工具箱提供的 c2d() 函数可以立即得出连续系统离散化的模型来，该函数的调用命令为

```
[F, G]=c2d(A, B, T);
```

其中显然地 (A, B) 为连续系统模型， T 为采样周期，而返回的 (F, G) 为离散时间系统模型。这样通过此函数的调用可以直接求出相应的离散时间系统状态方程模型来。

例 4.14　仍然考虑由例 4.2 中给出的连续状态方程模型，则调用 c2d() 函数可以立即得出在 $T=0.1$ 时的离散时间状态方程模型的相应矩阵为

```
>>  [F,G]=c2d(A,B,0.1)
F =  1.1915   -0.4455   -0.1013   -0.0421
```




```
     0.2008    0.6124   -0.1058   -0.0188
     0.0153   -0.0350    0.8849   -0.0905
     0.1147   -0.1622   -0.0197    0.9279
G =  0.3833    0.5527
     0.1906    0.3694
     0.1879    0.1764
     0.0048    0.1927
```

MATLAB 还提供了 c2dm() 函数来作类似的变换，其调用格式为

```
[F, G, Cd, Dd] = c2dm(A, B, C, D, T, 转换方法)
```

它与 c2d() 函数的区别在于，它可以允许用户自己选择变换的方法。MATLAB 提供的转换方法如表 4-1 所示。

表4-1　状态方程离散化的算法选项

选　项	说　明
'zoh'	假设对输入信号加一个零阶保持器
'foh'	假设对输入信号加一个一阶保持器
'tustin'	双线性变换方法 (Tustin 算法)
'prewarp'	改进的 Tustin 变换方法
'matched'	SISO 系统的零极点匹配法

对前面的例子采用两种不同的算法进行离散化，则可以得出下面的结果

```
>>  [f1,g1,c1,d1]=c2dm(A,B,C,D,0.1,'zoh')
f1 = 1.1915   -0.4455   -0.1013   -0.0421
     0.2008    0.6124   -0.1058   -0.0188
     0.0153   -0.0350    0.8849   -0.0905
     0.1147   -0.1622   -0.0197    0.9279
g1 = 0.3833    0.5527
     0.1906    0.3694
     0.1879    0.1764
     0.0048    0.1927
c1 =    0      0      0      1
        0      2      0      2
d1 =    0      0
        0      0
>>  [f2,g2,c2,d2]=c2dm(A,B,C,D,0.1,'tustin')
f2 = 1.1924   -0.4469   -0.1022   -0.0425
     0.2014    0.6115   -0.1064   -0.0191
     0.0156   -0.0355    0.8846   -0.0908
```




```
      0.1149   -0.1624   -0.0199    0.9279
g2 =  3.8358    5.5389
      1.9080    3.7018
      1.8804    1.7697
      0.0475    1.9279
c2 =  0.0057   -0.0081   -0.0010    0.0964
      0.0316    0.1449   -0.0126    0.1909
d2 =  0.0024    0.0964
      0.1955    0.5630
```

由这个例子可以看出，用零阶保持器的离散化得出的结果和直接调用 c2d() 函数所得的结果是一致的，它们都不对原系统的 C 和 D 矩阵作任何变动，而采用 Tustin 算法得出的矩阵 C 和 D 矩阵都和原系统不同，这就是说，在这样的变换下系统的状态变量选取得不同，所以最后得出的离散化模型描述也是不一致的。

离散时间系统连续化也可以由类似的方法得出，若仍令 $C=\widehat{C}, D=\widehat{D}$, 则可以证明其变换公式为

$$A=\frac{1}{T}\ln(F),\quad B=(F-I)^{-1}AG \tag{4.4.4}$$

MATLAB 的控制系统工具箱给出了一个离散时间状态方程连续化的函数 d2c()，其调用的格式为

```
[A, B]=d2c(F, G, T)
```

对例 4.14 变换出的离散时间状态方程连续化，则可以得出

```
>>  [A1, B1]=d2c(F, G, 0.1)
A1 =    2.2500   -5.0000   -1.2500   -0.5000
        2.2500   -4.2500   -1.2500   -0.2500
        0.2500   -0.5000   -1.2500   -1.0000
        1.2500   -1.7500   -0.2500   -0.7500
B1 =    4.0000    6.0000
        2.0000    4.0000
        2.0000    2.0000
        0.0000    2.0000
>>  norm(A-A1)
ans = 5.0746e-010
>>  norm(B-B1)
ans = 3.3832e-010
```

可见，变换后的模型和原来例 4.2 中所给出的形式是一致的，只是在结果上略有误差。

除了 d2c() 函数直接进行转换之外，MATLAB 的控制系统工具箱中还提供了和前面相应的 d2cm() 函数，该函数的调用格式为

```
[Ac, Bc, Cc, Dc] = c2dm(F, G, C, D, T, 转换方法)
```

其中转换方法可以使用的选项为 'zoh', 'tustin', 'prewarp' 和'matched'4 种，各个选项的意义和 c2dm() 函数中的几乎完全一致。

重新考虑前面由 Tustin 算法得出的离散时间系统状态方程模型，如果同样采用 'tustin' 选项进行反变换，则可以获得如下的结果

```
>>  [A1,B1,C1,D1]=d2cm(f2,g2,c2,d2,0.1,'tustin')
A1 =  2.2500   -5.0000   -1.2500   -0.5000
      2.2500   -4.2500   -1.2500   -0.2500
      0.2500   -0.5000   -1.2500   -1.0000
      1.2500   -1.7500   -0.2500   -0.7500
B1 =  4.0000    6.0000
      2.0000    4.0000
      2.0000    2.0000
      0.0000    2.0000
C1 =  0.0000    0.0000    0.0000    1.0000
      0.0000    2.0000    0.0000    2.0000
D1 = 1.0e-015 *
      0.0013    0.0139
     -0.0555   -0.1110
>>  [norm(A-A1) norm(B-B1) norm(C-C1) norm(D-D1)]
ans = 1.0e-014 *
    0.4030    0.2766    0.0445    0.0125
```

从转换的结果可以看出，除了在计算中产生了微小的误差之外，所得的结果和原始连续系统模型几乎完全一致。

MATLAB 的控制系统工具箱还提供了由带有时间延迟的连续系统转换成离散时间系统模型的转换函数 c2dt()，该函数的调用格式为

```
[F, G, Cd, Dd]=c2dt(A, B, C, T, tau)
```

其中 tau 为时间延迟常数 τ，该函数可以将

$$\dot{x}(t) = Ax(t) + Bu(t-\tau), \quad y(t) = Cx(t) \tag{4.4.5}$$

在采样周期 T 下离散化为

$$x_{k+1} = Fx_k + Gu_k, \quad y_k = C_d x_k + D_d u_k \tag{4.4.6}$$

例 4.15　假设系统的状态方程为

$$\dot{x}(t) = \begin{bmatrix} -10 & 1 & 0 & 0 \\ -35 & 0 & 1 & 0 \\ -50 & 0 & 0 & 1 \\ -24 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 7 \\ 24 \\ 24 \end{bmatrix} u(t-\tau), \quad y(t) = x_1$$

且时间延迟常数为 $\tau = 1$，如果想求出在采样周期为 $T = 0.2$ 时的离散时间系统模型，则可以使用如下的 MATLAB 命令



```
>> a=[-10,1,0,0; -35,0,1,0; -50,0,0,1; -24,0,0,0];
>> b=[1; 7; 24; 24]; c=[1,0,0,0];
>> [ad,bd,cd,dd]=c2dt(a,b,c,0.2,1)
ad = -0.0713  -2.8835  -3.6076  -1.6142 -21.3073        0        0        0        0
      0.0673   0.6013  -0.5295  -0.2448  -0.2744        0        0        0        0
      0.0102   0.1693   0.9583  -0.0195   4.8481        0        0        0        0
      0.0008   0.0183   0.1977   0.9989   5.2854        0        0        0        0
           0        0        0        0        0   1.0000        0        0        0
           0        0        0        0        0        0   1.0000        0        0
           0        0        0        0        0        0        0   1.0000        0
           0        0        0        0        0        0        0        0   1.0000
           0        0        0        0        0        0        0        0        0
 bd =    0
         0
         0
         0
         0
         0
         0
         0
         1
 cd =    1    0    0    0    0    0    0    0    0
 dd =    0
```

可见这样的转换对原系统的阶次作了很大的扩充，如果 τ 比采样周期 T 大得过多时，则不适于采用这种方法进行模型转换。

同样 MATLAB 还提供了带有时间延迟的离散时间状态方程转换成连续状态方程的函数 d2ct()，其调用方法与 c2dt() 很接近，在这里就不再赘述了。

MATLAB 的控制系统工具箱中并没有直接提供任何连续与脉冲传递函数之间的转换函数，用户可以通过编写自己的函数来作出相应的转换，下面将介绍一种间接的变换方法，即首先将要转换的传递函数转换成同一域下的状态方程模型，再对该状态方程作连续化或离散化，然后将所得出的结果再转换成同一域下的传递函数，这样就可以完成所需的转换。这里给出两个转换函数的清单

```
function [nd, dd] = tfc2d(nc, dc, T)
[a, b, c, d] = tf2ss(nc,dc);
[f, g] = c2d(a, b, T);
[nd, dd] = ss2tf(f, g, c, d, 1);

function [nc, dc] = tfd2c(nd, dd, T)
[a, b, c, d] = tf2ss(nd,dd);
[f, g] = d2c(a, b, T);
[nc, dc] = ss2tf(f, g, c, d, 1);
```




其中前一个函数的文件名为 tfc2d.m，它可以用来将一个已知连续系统传递函数模型 nc,dc 在采样周期 T 下离散化，得出脉冲传递函数 nd，dd，后一个函数的文件名为 tfd2c.m 将脉冲传递函数模型 nd，dd 在采样周期 T 下连续化成连续传递函数 nc，dc。注意，这两个变换函数只适合于单输入系统的变换。

4.5 状态方程的标准型转换

4.5.1 状态方程模型的相似变换

假设系统的状态方程模型由式 (4.1.8) 给出，若引入一个非奇异变换矩阵 $\mathcal{T}$，使得变换后的状态变量为 $z(t)=\mathcal{T}x(t)$，则可以将变换后的状态方程模型写出

$$\begin{cases}\dot{z}=A_t z+B_t u\\ y=C_t z+D_t u;\quad z(0)=\mathcal{T}x(0)\end{cases}\tag{4.5.1}$$

式中，变换后模型的各个矩阵可以写成

$$A_t=\mathcal{T}A\mathcal{T}^{-1},\quad B_t=\mathcal{T}B,\quad C_t=C\mathcal{T}^{-1},\quad D_t=D\tag{4.5.2}$$

这种变换称为相似变换，如果变换矩阵 $\mathcal{T}$ 为常数系数矩阵，则这种变换又称为线性变换，线性变换并不会改变系统的特征参数及结构。MATLAB 的控制系统工具箱给出了一个完成线性变换的函数 ss2ss()，该函数的调用方法为

```
[At, Bt, Ct, Dt] = ss2ss(A, B, C, D, T)
```

很显然，调用该函数之后就会将原来的状态方程模型 (A, B, C, D) 在变换矩阵 T 下相似地变换成 (At, Bt, Ct, Dt)。

从前面的叙述可以看出，变换后的系统状态方程模型形式取决于变换矩阵 $\mathcal{T}$ 的选择，亦即对变换矩阵进行不同的选择，则可以得出不同的状态方程表示形式。本节将分别介绍系统的可控型、可观测型以及 Jordan 标准型表示方式。

4.5.2 系统的可控性及可控标准型实现

考虑式 (4.1.8) 中给出的状态方程模型，如果按下面的方式定义一个矩阵 $\mathcal{W}_c$

$$\mathcal{W}_c=\begin{bmatrix}B & AB & A^2B & \cdots & A^{n-1}B\end{bmatrix}\tag{4.5.3}$$

且它为满秩矩阵，即 $\mathrm{rank}\{\mathcal{W}_c\}=n$，则系统 (A,B,C) 称为可控的。

假设 $B=[b_1\ \ b_2\ \ \cdots\ \ b_p]$，其中 b_i 为列向量，且

$$\sigma_i=\max_j\left\{\mathrm{rank}\{b_i\ \ Ab_i\ \ \cdots\ \ A^{j-1}b_i\}=j\right\}\tag{4.5.4}$$

可以按下面的方式构造一个方阵 S_c

$$S_c=\begin{bmatrix}b_1 & \cdots & A^{\sigma_1-1}b_1 & b_2 & \cdots & A^{\sigma_2-1}b_2 & \cdots & b_p & A^{\sigma_p-1}b_p & v_1 & \cdots & v_k\end{bmatrix}_{\text{取前 }n\text{ 列}}\tag{4.5.5}$$



式中 $v_i, i=1,\cdots,k$, $k=n-\text{rank}\{\mathcal{W}_c\}$, 且 v_i 为任意列向量，使得矩阵 S_c 为非奇异矩阵。这样可以容易地求出 S_c 矩阵的逆，假设其逆矩阵可以表示成

$$S_c^{-1}=\begin{bmatrix} l_1 \\ l_2 \\ \cdots \\ l_n \end{bmatrix} \tag{4.5.6}$$

则可以如下地构造一个可控型变换矩阵 $\mathcal{T}_c$

$$\mathcal{T}_c=\begin{bmatrix} l_1A^{\sigma_1-1} \\ \vdots \\ l_1 \\ l_{\sigma_1+1}A^{\sigma_2-1} \\ \vdots \\ l_{\sigma_1+1} \\ \vdots \end{bmatrix} \tag{4.5.7}$$

可控标准型的状态方程模型可以表示为

$$\begin{cases} \dot{x}_c=A_cx_c+B_cu \\ y=C_cx_c+D_cu; \quad x_c(0)=\mathcal{T}_cx(0) \end{cases} \tag{4.5.8}$$

式中

$$A_c=\mathcal{T}_cA\mathcal{T}_c^{-1}=\begin{bmatrix} A_{11} & \cdots & A_{1p} \\ A_{21} & \cdots & A_{2p} \\ \cdots & \cdots & \cdots \\ A_{p1} & \cdots & A_{pp} \end{bmatrix}, \quad B_c=\mathcal{T}_cB$$
$$C_c=C\mathcal{T}_c^{-1}, \quad D_c=D, \quad x_c=\mathcal{T}_cx \tag{4.5.9}$$

且

$$A_{ii}=\begin{bmatrix} -\alpha_{i1} & -\alpha_{i2} & \cdots & -\alpha_{i\sigma_1} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \tag{4.5.10}$$

例 4.16 考虑式 4.2 中给出的的状态方程模型，可以容易地看出 $\sigma_1=\text{rank}\left\{b_1 \ \ Ab_1 \ \ A^2b_1\right\}=3$ 。所以可以写出变换矩阵 S_c

$$S_c=\begin{bmatrix} b_1 & Ab_1 & A^2b_1 & b_2 \end{bmatrix}=\begin{bmatrix} 4 & -3.5 & 4.75 & 6 \\ 2 & -2 & 3.5 & 4 \\ 2 & -2.5 & 2.25 & 2 \\ 0 & 1 & -1 & 2 \end{bmatrix}$$

并可以得出其逆矩阵来

$$S_c^{-1}=\begin{bmatrix} 1.7857 & -1.7143 & -1.7571 & -0.5714 \\ 1.2857 & -0.7143 & -1.8571 & -0.5714 \\ 0.2857 & 0.2857 & -0.8571 & -0.5714 \\ -0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$$

取出该逆矩阵的第 1 和第 4 行 l_1, l_4，就可以容易地构造出系统的变换矩阵 T_c^{-1}

$$T_c = \begin{bmatrix} l_1 A^2 \\ l_1 A \\ l_1 \\ l_4 \end{bmatrix} = \begin{bmatrix} 0.1607 & 1.1250 & -1.4464 & -2.3036 \\ -0.8929 & 0.0357 & 1.7500 & 1.3214 \\ 1.7857 & -1.7143 & -1.3571 & -0.5714 \\ -0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$$

从而可以得出系统的可控标准型模型

$$A_c = T_c A T_c^{-1} = \begin{bmatrix} -2.5 & -2.5 & -1.5 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1.5 \end{bmatrix}, \quad B_c = T_c B = \begin{bmatrix} 0 & -2.0357 \\ 0 & 0.9286 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

除了前面介绍的可控标准型模型，还可以获得一种称作伴随形式的实现，首先任意地选择一个行向量 v，使得矩阵

$$\hat{T}_c = \begin{bmatrix} vA^{n-1} \\ \vdots \\ vA \\ v \end{bmatrix} \tag{4.5.11}$$

为非奇异矩阵，则伴随矩阵实现形式可以写成

$$\hat{A}_c = \hat{T}_c A \hat{T}_c^{-1} = \begin{bmatrix} -a_2 & -a_3 & \cdots & -a_n & -a_{n+1} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}, \quad \hat{B}_c = \hat{T}_c B, \quad \hat{C}_c = C\hat{T}_c^{-1} \tag{4.5.12}$$

式中 a_i 为矩阵 A 的特征方程系数，它满足

$$\det(sI - A) = s^n + a_2 s^{n-1} + a_3 s^{n-2} + \cdots + a_n s + a_{n+1} = 0 \tag{4.5.13}$$

对例 4.2 中给出的原始模型来说，当选择行向量 $v = [1 \ \ 0 \ \ 0 \ \ 0]$，则可以构造出一个非奇异的变换矩阵 $\hat{T}_c$，其表现形式为

$$\hat{T}_c = \begin{bmatrix} vA^3 \\ vA^2 \\ vA \\ v \end{bmatrix} = \begin{bmatrix} 13.3125 & -18.875 & -12.3125 & -5.75 \\ -7.125 & 11.5 & 5.125 & 1.75 \\ 2.25 & -5 & -1.25 & -0.5 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

系统的伴随矩阵标准型可以表示成

$$\hat{A}_c = \hat{T}_c A \hat{T}_c^{-1} = \begin{bmatrix} -4 & -6.25 & -5.25 & -2.25 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \hat{B}_c = \hat{T}_c B = \begin{bmatrix} -9.125 & -31.75 \\ 4.75 & 17 \\ -3.5 & -10 \\ 4 & 6 \end{bmatrix}$$

$$\hat{C}_c = C\hat{T}_c^{-1} = \begin{bmatrix} -0.5442 & -1.8367 & -2.1701 & -0.9592 \\ -1.0522 & -3.7732 & -5.1066 & -1.3878 \end{bmatrix}$$

如果 $\sigma_1 = n$，则向量 v 可以选择为 $v = l_n$，这样得出的变换模型与前面介绍的算法是一致的。



4.5.3 系统的可观测性及可观测标准型实现

如果按下面的方法构造出的一个矩阵 $\mathcal{W}_o$

$$\mathcal{W}_o = \begin{bmatrix} C \\ CA \\ CA^2 \\ \cdots \\ CA^{n-1} \end{bmatrix} \tag{4.5.14}$$

是满秩矩阵，亦即 $\text{rank}\{\mathcal{W}_o\} = n$，则系统 (A, B, C) 称为可观测的。

假设 $C = [c_1^T \quad c_2^T \quad \cdots \quad c_q^T]^T$，其中 c_i 为行向量，且

$$\gamma_i = \max_j \left\{ \text{rank} \begin{bmatrix} c_i \\ c_iA \\ \cdots \\ c_iA^{j-1} \end{bmatrix} = j \right\}, \quad i = 1, \cdots, q \tag{4.5.15}$$

则可以仿照前面的方法构造一个非奇异的矩阵 S_o

$$S_o = \begin{bmatrix} c_1 \\ \cdots \\ c_1A^{\gamma_1 - 1} \\ \cdots \\ c_qA^{\sigma_q - 1} \\ v_1 \\ \cdots \\ v_k \end{bmatrix} \text{取前 } n \text{ 行} \tag{4.5.16}$$

其中 $v_i, i = 1, \cdots, k$， $k = n - \text{rank}\{\mathcal{W}_o\}$ 为任意行向量，使得 S_o 矩阵为非奇异矩阵。求出 S_o 矩阵的逆并假设

$$S_o^{-1} = [\, l_1 \quad l_2 \quad \cdots \quad l_n \,] \tag{4.5.17}$$

式中 l_i 为列向量。按下面的方法构造一个变换矩阵 $\mathcal{T}_o$

$$\mathcal{T}_o = [\, A^{\gamma_1 - 1}l_1 \quad \cdots \quad l_1 \quad A^{\gamma_2 - 1}l_{\gamma_1 + 1} \quad \cdots \quad l_{\gamma_1 + 1} \quad \cdots] \tag{4.5.18}$$

则可以得出可观测标准型模型为

$$\begin{cases} \dot{\boldsymbol{x}}_o = A_o\boldsymbol{x}_o + B_o\boldsymbol{u} \\ \boldsymbol{y} = C_o\boldsymbol{x}_o + D_o\boldsymbol{u}; \quad \boldsymbol{x}_o(0) = \mathcal{T}_o\boldsymbol{x}(0) \end{cases} \tag{4.5.19}$$

且

$$A_o = \mathcal{T}_oA\mathcal{T}_o^{-1} = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ A_{21} & \cdots & A_{2q} \\ \cdots & \ddots & \cdots \\ A_{q1} & \cdots & A_{qq} \end{bmatrix}, \quad B_o = \mathcal{T}_oB$$
$$C_o = C\mathcal{T}_o, \quad D_o = \mathcal{T}_o^{-1}D\mathcal{T}_o, \quad \boldsymbol{x}_o = \mathcal{T}_o^{-1}\boldsymbol{x} \tag{4.5.20}$$

式中

$$A_{ii}=\begin{bmatrix}-\alpha_{i1} & 1 & 0 & \cdots & 0\\ -\alpha_{i2} & 0 & 1 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ -\alpha_{i\gamma_i} & 0 & 0 & \cdots & 0\end{bmatrix} \tag{4.5.21}$$

例 4.17　考虑例 4.2 中给出的模型，可见 $\gamma_1=3$，这样就可以建立起来一个 S_o 矩阵

$$S_o=\begin{bmatrix}c_1\\ c_1A\\ c_1A^2\\ c_2\end{bmatrix}=\begin{bmatrix}0 & 0 & 0 & 1\\ 1.25 & -1.75 & -0.25 & -0.75\\ -2.125 & 6.625 & 1.125 & 0.626\\ 0 & 2 & 0 & 2\end{bmatrix}$$

其逆矩阵可以写成

$$S_o^{-1}=\begin{bmatrix}-0.7143 & 1.2857 & 0.2857 & 0.75\\ -1 & 0 & 0 & 0.5\\ 0.4286 & 2.4286 & 1.4286 & 0.25\\ 1 & 0 & 0 & 0\end{bmatrix}$$

取出 S_o^{-1} 矩阵中的第 1 列和第 4 列 l_1, l_4，则可以构造出一个变换矩阵 $\mathcal{T}_o$

$$\mathcal{T}_o=\begin{bmatrix}A^2l_1 & Al_1 & l_1 & l_4\end{bmatrix}=\begin{bmatrix}-2.4643 & 2.3571 & -0.7143 & 0.75\\ -1.0714 & 1.8571 & -1 & 0.5\\ 1.1786 & -1.2143 & 0.4286 & 0.25\\ 0 & 0 & 1 & 0\end{bmatrix}$$

这样就可以写出可观测标准型模型

$$A_o=\mathcal{T}_o^{-1}A\mathcal{T}_o=\begin{bmatrix}-2.5 & 1 & 0 & 0\\ -2.5 & 0 & 1 & 0\\ -1.5 & 0 & 0 & 0\\ -4 & 0 & 0 & -1\end{bmatrix},\quad C_o=C\mathcal{T}_o=\begin{bmatrix}0 & 0 & 1 & 0\\ -2.1429 & 3.7143 & 0 & 1\end{bmatrix}$$

4.5.4 Jordan 标准型及其转换

假设系数矩阵 A 的特征值为

$$\lambda\{A\}=\lambda_1,\lambda_2,\cdots,\lambda_n \tag{4.5.22}$$

且其第 i 个特征向量 v_i 满足

$$Av_i=\lambda_i v_i,\ \ i=1,2,\cdots,n \tag{4.5.23}$$

这样就可以建立起来一个变换矩阵 Γ 使得

$$\Lambda=\Gamma^{-1}A\Gamma=\begin{bmatrix}J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_k\end{bmatrix} \tag{4.5.24}$$

式中 Λ 称为矩阵 A 的模态矩阵 (modal matrix)，而 J_i 称为 Jordan 子矩阵。如果矩阵 A 只包含单重实特征根，则系统的变换矩阵 Γ 可以由下面的方法构造出来

$$\Gamma=\begin{bmatrix}v_1 & v_2 & \cdots & v_n\end{bmatrix} \tag{4.5.25}$$

如果矩阵 A 包含有一对共轭复数特征根 $\lambda_{i,i+1}$，则系统的变换矩阵 Γ 可以是一个复数矩阵。但在这种情况下，我们往往希望从中建立起一个实数矩阵，这时

$$\Gamma = \begin{bmatrix} v_1 & v_2 & \cdots & \Re e\{v_i\} & \Im m\{v_{i+1}\} & \cdots & v_n \end{bmatrix} \tag{4.5.26}$$

如果原系统模型包含实数重根，则必须先构造一个到伴随矩阵的变换矩阵 $\widehat{\mathcal{T}}_c$。这样变换矩阵就可以定义为 $\Gamma = U\widehat{\mathcal{T}}_c$，式中 $U = \begin{bmatrix} U_1 & U_2 & \cdots & U_k \end{bmatrix}$。子矩阵 U_i 可以依照下列 3 种情况分别写出广义的 Vandermonde 矩阵

- 若 λ_i 为单重实数根，则

$$U_i = [1 \ \ \lambda_i \ \ \lambda_i^2 \ \ \cdots \ \ \lambda_i^{n-1}]^T, \quad J_i = \lambda_i \tag{4.5.27}$$

- 若 λ_i 为 m_i 重的实数根，则子矩阵 U_i 和 J_i 可以如下地建立起来

$$U_i = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \lambda_i & 1 & 0 & \cdots & 0 \\ \lambda_i^2 & 2\lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_i^{n-1} & \dfrac{d}{d\lambda_i}(\lambda_i^{n-1}) & \dfrac{1}{2!}\dfrac{d^2}{d\lambda_i^2}(\lambda_i^{n-1}) & \cdots & \dfrac{1}{(m_i-1)!}\dfrac{d^{m_i-1}}{d\lambda_i^{m_i-1}}(\lambda_i^{n-1}) \end{bmatrix} \tag{4.5.28}$$

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_i \end{bmatrix} \tag{4.5.29}$$

- 若 $\lambda_{i,i+1} = \sigma_i \pm j\omega_i$ 为一对共轭复数根，则相应的子矩阵可以如下建立起来

$$U_i = \begin{bmatrix} 1 & 0 \\ \sigma_i & \omega_i \\ \vdots & \vdots \\ \Re e\{\lambda_i^{n-1}\} & \Im m\{\lambda_i^{n-1}\} \end{bmatrix}, \quad J_i = \begin{bmatrix} \sigma_i & \omega_i \\ -\omega_i & \sigma_i \end{bmatrix} \tag{4.5.30}$$

例 4.18 再考虑由例 4.2 中给出的系统模型，A 矩阵的特征值可以求出

$$\lambda\{A\} = -1.5, \ -1.5, \ -0.5 \pm j0.866$$

这样就可以写出广义的 Vandermonde 矩阵 U

$$U = \begin{bmatrix} 1 & 0 & 1 & 0 \\ \lambda_1 & 1 & \Re e\{\lambda_3\} & \Im m\{\lambda_3\} \\ \lambda_1^2 & 2\lambda_1 & \Re e\{\lambda_3^2\} & \Im m\{\lambda_3^2\} \\ \lambda_1^3 & 3\lambda_1^2 & \Re e\{\lambda_3^3\} & \Im m\{\lambda_3^3\} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ -1.5 & 1 & -0.5 & 0.866 \\ 2.25 & -3 & -0.5 & -0.866 \\ -3.375 & 6.75 & 1 & 0 \end{bmatrix}$$

利用例 4.2 中得出的变换矩阵 $\widehat{\mathcal{T}}_c$，可以得出一个变换矩阵 Γ 为

$$\Gamma = U\widehat{\mathcal{T}}_c = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0.6667 & -0.1111 & 0.5 & -0.2887 \\ 0.3333 & -0.2222 & 0 & 0.5774 \\ 0 & -0.3333 & 0.5 & -0.2887 \end{bmatrix}$$

这样就可以求出系统的 Jordan 标准型模型为

$$A_j=\Gamma^{-1}A\Gamma=\begin{bmatrix}-1.5 & 1 & 0 & 0\\ 0 & -1.5 & 0 & 0\\ 0 & 0 & -0.5 & 0.866\\ 0 & 0 & -0.866 & -0.5\end{bmatrix},\quad B_j=\Gamma^{-1}B=\begin{bmatrix}3 & 4\\ 0 & -3\\ 1 & 2\\ 1.7321 & 0\end{bmatrix}$$

$$C_j=C\Gamma=\begin{bmatrix}0 & -0.3333 & 0.5 & -0.2887\\ 1.33333 & -0.8889 & 2 & -1.1547\end{bmatrix}$$

MATLAB 的控制系统工具箱提供了一个系统标准型转换的函数 canon()，该函数的调用格式为

```
[As, Bs, Cs, Ds, T]=canon(A, B, C, D, 类型)
```

其中 A，B，C，D 为原系统状态方程模型，而返回的 As，Bs，Cs，Ds 为指定的标准型的状态方程模型，T 为转换矩阵。这里的类型可以有两个选项，'modal' 和 'companion'(前 3 个字符一致即可)，分别可以给出对角标准型和伴随矩阵型，但这里用到的算法存在问题，例如在含有重根时 'modal' 标准型得不出标准的 Jordan 型，而是对角形 (变换矩阵 T 的值将取得特别大)。在系统 B 矩阵的第 1 列构成的子系统不完全能控时将得不出系统的伴随矩阵标准型，所以在使用此函数时应该引起注意。

考虑例 4.2 中给出的原系统模型，如果调用 canon() 函数来分别求出 Jordan 标准型和伴随标准型，则可以由下面的命令直接完成

```
>>  [As, Bs, Cs, Ds, T]=canon(A, B, C, D, 'mod')
As = -0.5000    0.8660    0.0000    0.0000
     -0.8660   -0.5000    0.0000    0.0000
      0.0000    0.0000   -1.5000    0.0000
      0.0000    0.0000    0.0000   -1.5000
Bs =  1.0e+008 *
      0.0000    0.0000
      0.0000    0.0000
      0.0000   -3.1888
      0.0000    0.0000
Cs =  0.4074    0.0259    0.0000    0.0000
      1.6297    0.1036    0.0000   -1.0690
Ds =  0     0
      0     0
T =  1.0e+008 *
      0.0000    0.0000    0.0000    0.0000
      0.0000    0.0000    0.0000    0.0000
      1.5944   -1.5944   -1.5944   -1.5944
      0.0000    0.0000    0.0000    0.0000
>>  format long; As
As = -0.50000000000000  0.86602540378444  0.00000000000000  0.00000000000000
     -0.86602540378444 -0.50000000000000  0.00000000000000  0.00000000000000
      0.00000004950562 -0.00000003219520 -1.49999999999998  0.00000001309307
```




```
      -0.00000000000018  0.00000000000007 -0.00000001173392 -1.49999999999991
>>  As(3,3)-As(4,4)
ans = -6.994405055138486e-014
>>  [As, Bs, Cs, Ds, T]=canon(A, B, C, D, 'com')
??? Error using ==> canon
System must be controllable from first input.
```

其实这里得出的 Jordan 标准型是错误的，由于求根算法本身的原因，MATLAB 不认为得出的第 3 个根和第 4 个根是相同的，虽然它们都约等于 -1.5，但实质上得出的根有 10^{-14} 的差异，从而会产生错误的结果。在后面的例子中因为原系统对第 1 输入信号并不完全可控，所以不能得出系统的伴随标准型。

4.6 状态方程的最小实现

在介绍线性系统的最小实现之前，有必要先叙述系统的 Kalman 分解的概念。可以证明，由式 (4.1.8) 给出的控制系统模型可以等效地变换成下面的规范形式 [5]

$$\dot{z}=\left[\begin{array}{c|c|c|c}\widehat{A}_{c,o} & 0 & \widehat{A}_{1,2} & 0\\ \hline \widehat{A}_{2,1} & \widehat{A}_{c,\bar{o}} & \widehat{A}_{2,3} & \widehat{A}_{2,4}\\ \hline 0 & 0 & \widehat{A}_{\bar{c},o} & 0\\ \hline 0 & 0 & \widehat{A}_{4,3} & \widehat{A}_{\bar{c},\bar{o}}\end{array}\right]z+\left[\begin{array}{c}\widehat{B}_{c,o}\\ \hline \widehat{B}_{c,\bar{o}}\\ \hline 0\\ \hline 0\end{array}\right]u,\quad y=\left[\begin{array}{c|c|c|c}\widehat{C}_{c,o} & 0 & \widehat{C}_{\bar{c},o} & 0\end{array}\right]z \tag{4.6.1}$$

式中 $(\widehat{A}_{c,o},\widehat{B}_{c,o},\widehat{C}_{c,o})$ 为既可控又可观测的子空间，$(\widehat{A}_{c,\bar{o}},\widehat{B}_{c,\bar{o}},0)$, $(\widehat{A}_{\bar{c},o},0,\widehat{C}_{c,\bar{o}})$, $(\widehat{A}_{\bar{c},\bar{o}},0,0)$ 分别为可控但不可观测、可观测但不可控以及既不可控又不可观测的子空间。上面的分解方式又称为 Kalman 分解形式。Kalman 分解的示意性框图表示如图 4-5 所示。

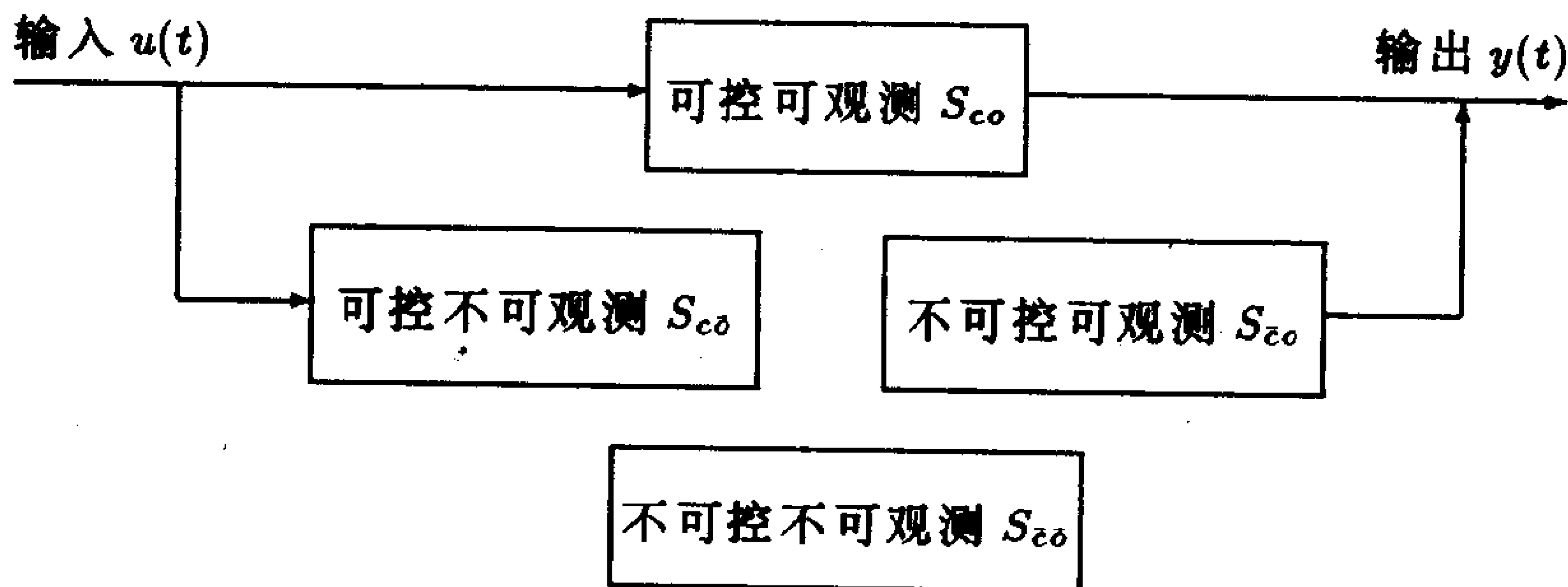


图 4-5　Kalman 分解的示意图

在 Kalman 分解形式中，子空间 $(\widehat{A}_{c,o},\widehat{B}_{c,o},\widehat{C}_{c,o})$ 称为原始系统的最小实现形式，亦即系统的最小实现模型既可控又可观测。从传递函数角度来说，将系统所含有的相同零点和极点对消之后，所得出的模型为最小实现形式。

系统的最小实现可以由下面的三步法则来求出

- 首先找出一个变换矩阵 $\mathcal{T}_c^{-1}$ 以便得出系统可控标准型，然后将不可控部分分离出来

$$A_c = \mathcal{T}_c^{-1} A \mathcal{T}_c = \begin{bmatrix} \widehat{A}_c & 0 \\ 0 & \widehat{A}_{\bar{c}} \end{bmatrix}, \quad B_c = \mathcal{T}_c^{-1} B = \begin{bmatrix} \widehat{B}_c \\ 0 \end{bmatrix}, \quad C_c = C\mathcal{T}_c = \begin{bmatrix} \widehat{C}_c & \widehat{C}_{\bar{c}} \end{bmatrix} \tag{4.6.2}$$

- 找出变换矩阵 $\widehat{\mathcal{T}}_o$ 使得可控子系统 $(\widehat{A}_c, \widehat{B}_c, \widehat{C}_c)$ 可以被分解出可观测部分

$$\widehat{A}_o = \widehat{\mathcal{T}}_o^{-1} \widehat{A}_c \widehat{\mathcal{T}}_o = \begin{bmatrix} \widehat{A}_{c,o} & 0 \\ 0 & \widehat{A}_{c,\bar{o}} \end{bmatrix}, \quad \widehat{B}_o = \widehat{\mathcal{T}}_o^{-1} \widehat{B}_c = \begin{bmatrix} \widehat{B}_{c,o} \\ 0 \end{bmatrix}, \quad \widehat{C}_o = \widehat{C}_c \widehat{\mathcal{T}}_o = \begin{bmatrix} \widehat{C}_{c,o} & \widehat{C}_{c,\bar{c}} \end{bmatrix} \tag{4.6.3}$$

然后构造一个矩阵 $\widetilde{\mathcal{T}}_o^{-1} = \begin{bmatrix} \widehat{\mathcal{T}}_o^{-1} & 0 \\ 0 & I_{n-\mathrm{rank}\{\widehat{A}_c\}} \end{bmatrix}$.

- 构造变换矩阵 $\mathcal{T}^{-1} = \widetilde{\mathcal{T}}_o^{-1} \mathcal{T}_c^{-1}$，对系统进行变换，可得出 $(\widehat{A}_{c,o}, \widehat{B}_{c,o}, \widehat{C}_{c,o})$ 即为最小实现形式。

例 4.19 考虑下面给出的一个 4 阶模型

$$\dot{x} = \begin{bmatrix} -5 & 8 & 0 & 0 \\ -4 & 7 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & -2 & 6 \end{bmatrix} x + \begin{bmatrix} 4 \\ -2 \\ 2 \\ 1 \end{bmatrix} u, \quad y = [2 \ \ -2 \ \ -2 \ \ 2]x$$

可见 $\sigma_1 = 3$，选择一个向量 $v_1 = [0 \ \ 0 \ \ 0 \ \ 1]^T$，这样可控变换的变换矩阵为

$$\mathcal{T}_c^{-1} = \begin{bmatrix} 0.0451 & -0.0764 & -0.1667 & 0 \\ 0.0799 & -0.1736 & 0 & -0.6667 \\ 0.2951 & -0.5764 & 1.3333 & -4 \\ 0 & 0 & -0.5 & 1 \end{bmatrix}$$

从而可以写出系统的可控标准型

$$A_c = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -6 & -1 & 4 & -3.3333 \\ \hline 0 & 0 & 0 & 4 \end{array}\right], \quad B_c = \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 0 \end{array}\right], \quad C_c = \left[\begin{array}{ccc|c} 78 & -56 & 10 & 4.6667 \end{array}\right]$$

考虑一下可控子模型 $(\widehat{A}_c, \widehat{B}_c, \widehat{C}_c)$，可以得出 $\gamma_1 = 2$，引入一个向量 $v_1 = [1 \ 0 \ 0]$，则可以得出一个变换矩阵

$$\widehat{\mathcal{T}}_o^{-1} = \begin{bmatrix} 0 & -0.0463 & 1 \\ -0.0463 & -0.2593 & 3 \\ -0.2593 & -0.9907 & 9 \end{bmatrix}$$

这样就可以获得一个变换矩阵 $\mathcal{T}^{-1}$，使得

$$\mathcal{T}^{-1} = \begin{bmatrix} \widehat{\mathcal{T}}_o^{-1} & 0 \\ 0 & 1 \end{bmatrix} \mathcal{T}_c^{-1} = \begin{bmatrix} -4 & 4 & -11.6667 & 21.3333 \\ 2 & -2 & 0.3333 & -2.6667 \\ 0.1377 & -0.1690 & 0.1512 & -0.1235 \\ 0 & 0 & -0.5 & 1 \end{bmatrix}$$

由此可以得出系统的 Kalman 分解形式

$$\dot{z} = \left[\begin{array}{cc|c|c} 0 & 2 & 0 & 86.6667 \\ 1 & 1 & 0 & -33.3333 \\ \hline 0 & -0.2130 & 3 & -1.5432 \\ \hline 0 & 0 & 0 & 4 \end{array}\right] z + \left[\begin{array}{c} -26 \\ 10 \\ \hline 0.4630 \\ \hline 0 \end{array}\right] u, \quad y = z_2 + 4.6667 z_4$$



取出其中的既可控又可观测的子空间 $(\widehat{A}_{c,o}, \widehat{B}_{c,o}, \widehat{C}_{c,o})$, 则可以得出系统的最小实现为

$$\dot{\hat{z}} = \begin{bmatrix} 0 & 2 \\ 1 & 1 \end{bmatrix} \hat{z} + \begin{bmatrix} -26 \\ 10 \end{bmatrix} u, \quad y = \hat{z}_2$$

这一问题还可以由另外的方法得出，首先得出系统的传递函数并进行分解因式得出

$$G(s) = C(sI - A)^{-1}B = \frac{2(s-3)(s-4)(s-5)}{(s+1)(s-2)(s-3)(s-4)}$$

对消掉共同的零极点 $s=3$ 与 $s=4$, 则可以得出最小实现的传递函数 $\widehat{G}(s)$ 为

$$\widehat{G}(s) = \frac{2(s-5)}{(s+1)(s-2)} = \frac{2s-10}{s^2-s-2}$$

可以证明其结果和前面得出的是完全一致的。

利用 MATLAB 控制系统工具箱提供的 minreal() 函数也可以直接求出一个给定状态方程模型的最小实现，该函数的调用格式为

```
[Am, Bm, Cm, Dm]=minreal(A, B, C, D, tol);
```

其中 A, B, C, D 为原模型的状态方程矩阵，而 tol 为用户任意指定的误差限，如果省略此参数，则会自动地取作 eps 。调用此函数之后，就会自动地返回一个最小实现的状态方程模型 Am, Bm, Cm, Dm 。首先给 A, B, C, D 矩阵赋值，再调用此函数，则可以得出系统的最小实现模型为[1)]

```
>>  A=[-5,8,0,0; -4,7,0,0; 0,0,0,4; 0,0,-2,6];
>>  B=[4; -2; 2; 1]; C=[2,-2,-2,2]; D=0;
>>  [Am,Bm,Cm,Dm]=minreal(A,B,C,D)
2 states removed
Am =   1.7273     0.8624
       0.8624    -0.7273
Bm = -3.4112
       3.3710
Cm =   0.0000     2.9665
Dm =   0
```

如果原模型由传递函数形式 num, den 给出，则可以直接调用 minreal() 函数来获得最小实现的传递函数 NUMm, DENm 。这时的调用格式为

```
[NUMm, DENm] = minreal(num, den, tol)
```

考虑前面的例子，可以容易地得出系统的传递函数模型，然后由传递函数模型直接进行最小实现运算

```
>>  [num, den]=ss2tf(A, B, C, D, 1)
num =   0    2.0000    36.0000 -366.0000   760.0000
den =   1    -18    147   -498    536
>>  [NUMm, DENm] = minreal(num, den)
2 pole-zeros cancelled
NUMm =  0     2.0000   -10.0000
DENm =  1.0000    -1.0000    -2.0000
```

可见这时获得的传递函数模型和前面得出的是一致的。

[1)]注意：这样得出的最小实现模型可能和前面算法的不同，但它们是完全等效的。

4.7 状态方程的均衡实现

在讨论均衡实现之前，首先考虑下面一个著名的例子 [10]

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{bmatrix}-1 & 0\\ 0 & -2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}10^{-6}\\ 10^{6}\end{bmatrix}u,\quad y(t)=[10^6\ \ 10^{-6}]\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

可见，B 矩阵中的第 1 个元素的值比第 2 个值小得多，而 C 矩阵的情况也是类似的。这样在数值运算中由于舍入等情况可能将带来误差，因为在数值运算中比较小的值往往会被截断。如果我们引入一对新状态变量 $z_1=10^6x_1$ 和 $z_2=10^{-6}x_2$ 来重新对原系统作变换以改变其标度，则原始系统的模型可以变换成下面的形式

$$\begin{bmatrix}\dot{z}_1\\ \dot{z}_2\end{bmatrix}=\begin{bmatrix}-1 & 0\\ 0 & -2\end{bmatrix}\begin{bmatrix}z_1\\ z_2\end{bmatrix}+\begin{bmatrix}1\\ 1\end{bmatrix}u,\quad y(t)=[1\ \ 1]\begin{bmatrix}z_1\\ z_2\end{bmatrix}$$

这样的变换后的系统称为均衡实现的系统。可见在这种变换下只改变了各个状态变量的内部坐标标度，使得 B 矩阵及 C 矩阵的相应元素之间相差不再像原来那样悬殊。

对原系统进行均衡实现，首先可以定义可控和可观测 Gram 矩阵 (controllability gramian and observability gramian)

$$\mathcal{G}_c=\int_0^\infty \mathrm{e}^{At}BB^T\mathrm{e}^{A^Tt}\mathrm{d}t,\quad \mathcal{G}_o=\int_0^\infty \mathrm{e}^{A^Tt}C^TC\mathrm{e}^{At}\mathrm{d}t \tag{4.7.1}$$

可以证明 $\mathcal{G}_c$ 与 $\mathcal{G}_o$ 均为对称的半正定矩阵，并分别满足下面的 Lyapunov 方程

$$A\mathcal{G}_c+\mathcal{G}_cA^T=-BB^T,\quad A^T\mathcal{G}_o+\mathcal{G}_oA=-C^TC \tag{4.7.2}$$

这时总存在一个矩阵 $\mathcal{T}_b$，它可以将不对称的稳定原始系统变换成下面的形式

$$\begin{cases}\dot{z}=A_bz+B_bu\\ y=C_bz+D_bu;\quad z(0)=\mathcal{T}_bx(0)\end{cases} \tag{4.7.3}$$

式中

$$z=\mathcal{T}_bx,\quad A_b=\mathcal{T}_b^{-1}A\mathcal{T}_b,\quad B_b=\mathcal{T}_b^{-1}B,\quad C_b=C\mathcal{T}_b,\quad D_b=D \tag{4.7.4}$$

这一系统满足 Lyapunov 方程

$$A_b\Sigma+\Sigma A_b^T=-B_bB_b^T,\quad A_b^T\Sigma+\Sigma A_b=-C_b^TC_b \tag{4.7.5}$$

其中 Σ 为对角矩阵，可见，均衡变换后系统的可控与可观测 Gram 矩阵是相同的对角矩阵且均为 Σ。这时得出的系统状态方程模型 (A_b,B_b,C_b,D_b) 称作内部均衡 (internally balanced) 的系统。在实际应用中我们可以采用下面的方式来构造变换矩阵 $\mathcal{T}_b$

- 求解式 (4.7.2) 中给出的 Lyapunov 方程来获得 $\mathcal{G}_c$ 和 $\mathcal{G}_o$ 矩阵
- 对 $\mathcal{G}_o$ 矩阵进行 Cholesky 分解，使得 $\mathcal{G}_o=R^TR$



- 进行下面的奇异值分解： $R\mathcal{G}_c R^T = U\Sigma^2 U^T$
- 由下式求出变换矩阵 $\mathcal{T}_b$

$$\mathcal{T}_b^{-1} = \Sigma^{-1/2} U^T R \tag{4.7.6}$$

上述的计算方法可以简单地由 MATLAB 实现出来，假设给出了系统的 (A, B, C) 模型，则均衡实现的变换矩阵 T_b 可以由下面的程序计算出来

```
Gc=lyap(A,B*B');
Go=lyap(A',C'*C);
R=chol(Go);
[U,Sigma,V]=svd(R*Gc*R');
Sigma=sqrt(Sigma);
Tb=diag(diag(Sigma).^(-1/2))*U'*R;
```

其中调用了两次 lyap() 函数，它们分别用来求解相应的 Lyapunov 方程，这里还调用了 chol() 函数来对 $\mathcal{G}_o$ 进行 Cholesky 分解。可见，看起来比较难求出的均衡变换矩阵可以简洁地由几条 MATLAB 语句就求出来了。

例 4.20 再考虑一下例 4.2 中给出的系统模型，求解 Lyapunov 方程可以容易地获得可控 Gram 矩阵 $\mathcal{G}_c$ 和可观测 Gram 矩阵 $\mathcal{G}_o$

$$\mathcal{G}_c = \begin{bmatrix} 20.7627 & 12.5182 & 5.5479 & 6.3809 \\ 12.5182 & 7.8655 & 2.9221 & 4.3387 \\ 5.5479 & 2.9221 & 2.4403 & 0.8755 \\ 6.3809 & 4.3387 & 0.8755 & 2.8860 \end{bmatrix}, \quad \mathcal{G}_o = \begin{bmatrix} 8.4657 & -0.1472 & -0.3369 & 3.6780 \\ -10.4716 & 14.2517 & -0.2230 & -3.4857 \\ -0.3369 & -0.2230 & 0.7081 & -0.7411 \\ 3.6780 & -3.4857 & -0.7411 & 3.0313 \end{bmatrix}$$

通过前面给出的变换也可以容易地求出 R, U 和 Σ 矩阵

$$R = \begin{bmatrix} 2.9096 & -3.5990 & -0.1158 & 1.2641 \\ 0 & 1.1397 & -0.5613 & 0.9335 \\ 0 & 0 & 0.6161 & -0.1148 \\ 0 & 0 & 0 & 0.7408 \end{bmatrix}, \quad U = \begin{bmatrix} 0.7655 & -0.5896 & 0.2574 & -0.0056 \\ 0.6149 & 0.7001 & -0.2186 & 0.2896 \\ 0.0748 & -0.3122 & -0.9402 & -0.1136 \\ 0.1740 & 0.2542 & 0.0442 & -0.9504 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 6.4451 & & & \\ & 2.1938 & & \\ & & 0.3958 & \\ & & & 0.1290 \end{bmatrix}$$

由这些矩阵可以立即求出均衡系统变换矩阵 $\mathcal{T}_b^{-1}$ 为

$$\mathcal{T}_b^{-1} = \begin{bmatrix} 0.8773 & -0.8092 & -0.1527 & 0.6547 \\ -1.1583 & 1.9715 & -0.3491 & 0.0893 \\ 1.1906 & -1.8687 & -0.7732 & 0.4166 \\ -0.0451 & 0.9747 & -0.6456 & -1.1905 \end{bmatrix}$$

均衡变换后的系统模型为

$$\begin{cases} \dot{z} = \begin{bmatrix} -0.9078 & -0.9118 & 0.5703 & 0.1146 \\ 0.4561 & -0.4789 & 0.7617 & 0.1720 \\ 0.5843 & -0.2691 & -1.7208 & -0.5804 \\ -0.0993 & 0.3007 & 0.7033 & -0.8925 \end{bmatrix} z + \begin{bmatrix} 1.5856 & 3.0312 \\ -1.3885 & 0.4165 \\ -0.5213 & -1.0442 \\ 0.4779 & -0.0437 \end{bmatrix} u \\ \hat{y}(t) = \begin{bmatrix} 0.5963 & 0.5082 & 0.0376 & -0.4608 \\ 3.3685 & 1.3576 & -1.1665 & -0.1340 \end{bmatrix} z \end{cases}$$

对均衡变换矩阵 T_b 的计算还有更简单的算法，例如由文献 [6] 给出的算法。在 MATLAB 的控制系统工具箱下给出了一个基于该算法的均衡变换函数 balreal()，该函数的调用格式为

```
[Ab, Bb, Cb, G, T] = balreal(A, B, C)
```

这一函数由给定的状态方程模型 (A, B, C) 求出均衡实现的模型 (A_b, B_b, C_b)，并求出均衡系统的 Gram 矩阵 G (即前面算法中使用的 Σ 矩阵) 和变换矩阵 T 。

考虑前面给出的系统模型，如果调用 balreal() 函数则将直接得出系统的均衡实现模型及变换矩阵

```
>>  [Ab,Bb,Cb,G,T] = balreal(A,B,C)}
Ab = -0.9078    0.9118    0.5703    0.1146
     -0.4561   -0.4789   -0.7617   -0.1720
      0.5843    0.2691   -1.7208   -0.5804
     -0.0993   -0.3007    0.7033   -0.8925
Bb =  1.5856    3.0312
      1.3885   -0.4165
     -0.5213   -1.0442
      0.4779   -0.0437
Cb =  0.5963   -0.5082    0.0376   -0.4608
      3.3685   -1.3576   -1.1665   -0.1340
G =   6.4451    2.1938    0.3958    0.1290
T =   1.7714    0.3360   -0.7665    0.6806
      1.0880   -0.1706   -0.6208    0.3938
      0.4194    0.6560   -0.9530   -0.1521
      0.5963   -0.5082    0.0376   -0.4608
```

由于均衡变换矩阵的选择略有不同，所有用 MATLAB 控制系统工具箱中函数变换的结果和前面得出的不完全一致，但细心观察两种结果就会发现它们都满足均衡实现的条件，由此可见，系统的均衡实现模型并不是唯一的。

均衡实现实际上是一种系统状态的变换，由于它对模型的表示方法做了特殊的规范化处理，所以在提高数值稳定性角度的意义是相当明显的。另外均衡实现在模型降阶技术中也是很有意义的，在下面一节中将简要介绍均衡实现模型降阶方法。

4.8 控制系统辨识与降阶技术

4.8.1 连续系统的模型辨识

在控制系统研究中经常会遇到这样的问题，即用户没有办法从物理上得出所研究系统的数学模型，但可以通过适当的实验手段测试出系统的某种响应信息，如可以通过频率响应测试仪来测试出系统的频率响应数据，或通过数据采集系统来测试出系统时间响应的输入与输出数据，有了系统的某种响应数据，就可以根据它来获得系统的数学模



型，这种获得系统模型的过程称为系统辨识。

由频率响应数据来辨识系统模型的想法起源于 Levy 的复数曲线拟合方法，对下面给定的离散频率采样点 $\{\omega_i\}$，$i=1,2,\cdots,N$ 假定已经测试出系统的频率响应数据为 $\{P_i,Q_i\}$，其中 $\widehat{G}(j\omega_i)=P_i+Q_ij$，对连续系统传递函数

$$G(s)=\frac{\beta_0+\beta_1s+\beta_2s^2+\cdots+\beta_rs^r}{1+\alpha_1s+\alpha_2s^2+\cdots+\alpha_ms^m} \tag{4.8.1}$$

来说，可以简单地得出

$$G(j\omega)=\frac{\beta_0+\beta_1j\omega+\cdots+\beta_r(j\omega)^r}{1+\alpha_1j\omega+\cdots+\alpha_m(j\omega)^m}=\frac{B_1(\omega)+jB_2(\omega)}{A_1(\omega)+jA_2(\omega)} \tag{4.8.2}$$

其中

$$\begin{aligned}
B_1(\omega)&=\beta_0-\beta_2\omega^2+\cdots=\sum_{i=0}^{[r/2]}(-1)^i\beta_{2i}\omega^{2i}\\
B_2(\omega)&=\beta_1\omega-\beta_3\omega^3+\cdots=\sum_{i=0}^{[(r-1)/2]}(-1)^i\beta_{2i+1}\omega^{2i+1}\\
A_1(\omega)&=\alpha_0-\alpha_2\omega^2+\cdots=\sum_{i=0}^{[m/2]}(-1)^i\alpha_{2i}\omega^{2i}\\
A_2(\omega)&=\alpha_1\omega-\alpha_3\omega^3+\cdots=\sum_{i=0}^{[m/2]}(-1)^i\alpha_{2i+1}\omega^{2i+1}
\end{aligned} \tag{4.8.3}$$

则可以由下面几个表达式定义一些中间数值

$$\begin{aligned}
\lambda_i&=\sum_{k=1}^{N}\omega_k^i, & S_i&=\sum_{k=1}^{N}\omega_k^iP_k\\
T_i&=\sum_{k=1}^{N}\omega_k^iQ_k, & U_i&=\sum_{k=1}^{N}\omega_k^i[P_k^2+Q_k^2]
\end{aligned} \tag{4.8.4}$$

并引入拟合的性能指标

$$J=\sum_{k=1}^{N}|\,D(j\omega_k)e(j\omega_k)\,|^2 \tag{4.8.5}$$

式中 $e(j\omega_k)=G(j\omega_k)-\widehat{G}(j\omega_k)$ 为频率拟合的误差，且 $G(j\omega_k)$ 为从辨识出来的模型计算出来的频率响应数据，并假定 $\{D(j\omega_k)\}$ 为加权系数，则可以看出，如果对性能指标取各个参数的导数，并假定它们为 0

$$\frac{\partial J}{\partial\beta_i}=0,\ i=0,\cdots,r,\ \text{且}\ \frac{\partial J}{\partial\alpha_i}=0,\ i=1,\cdots,m \tag{4.8.6}$$

则可以获得性能指标 J 的最小值。由式 (4.8.6) 可以推导出下面的线性代数方程

$$\Gamma X=B \tag{4.8.7}$$

式中

$$\Gamma=\begin{bmatrix}\lambda_0 & 0 & -\lambda_2 & 0 & \lambda_4 & \cdots & T_1 & S_2 & -T_3 & -S_4 & T_5 & \cdots\\ 0 & \lambda_2 & 0 & -\lambda_4 & 0 & \cdots & -S_2 & T_3 & S_4 & -T_5 & -S_6 & \cdots\\ -\lambda_2 & 0 & \lambda_4 & 0 & -\lambda_6 & \cdots & -T_3 & -S_4 & T_5 & S_6 & -T_7 & \cdots\\ 0 & -\lambda_4 & 0 & \lambda_6 & 0 & \cdots & S_4 & -T_5 & -S_6 & T_7 & S_8 & \cdots\\ \lambda_4 & 0 & -\lambda_6 & 0 & \lambda_8 & \cdots & T_5 & S_6 & -T_7 & -S_8 & T_9 & \cdots\\ \vdots & & & & & \ddots & & & & & & \\ T_1 & -S_2 & -T_3 & S_4 & T_5 & \cdots & U_2 & 0 & -U_4 & 0 & U_6 & \cdots\\ S_2 & T_3 & -S_4 & -T_5 & S_6 & \cdots & 0 & U_4 & 0 & -U_5 & 0 & \cdots\\ -T_3 & S_4 & T_5 & -S_6 & -T_7 & \cdots & -U_4 & 0 & U_6 & 0 & -U_8 & \cdots\\ -S_4 & -T_5 & S_6 & T_7 & -S_8 & \cdots & 0 & -U_6 & 0 & U_8 & 0 & \cdots\\ T_5 & -S_6 & -T_7 & S_8 & T_9 & \cdots & U_5 & 0 & -U_8 & 0 & U_{10} & \cdots\\ \vdots & & & & & & & & & & & \end{bmatrix} \tag{4.8.8}$$

且

$$\begin{aligned}X^T&=\begin{bmatrix}\beta_0 & \beta_1 & \beta_2 & \beta_3 & \beta_4 & \cdots & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \cdots\end{bmatrix}\\ B&=\begin{bmatrix}S_0 & T_1 & -S_2 & -T_3 & S_4 & \cdots & 0 & U_2 & 0 & -U_4 & 0 & \cdots\end{bmatrix}\end{aligned} \tag{4.8.9}$$

可见，有了式 (4.8.4) 中定义的各个参数之后，待辨识系统的传递函数模型可以通过求解线性代数方程的方法容易地求出。依照前面给出的算法可以容易地编写出如下的 MATLAB 函数 `freq2tf()`，并利用它直接进行连续传递函数的辨识。

```
function [num,den]=freq2tf(H, w, nA, nB)
P=real(H); Q=imag(H); M=length(w);
for i=1:2:nA+nB+1
   L(i)=sum(w.^(i-1));  S(i)=sum(w.^(i-1).*P);
   S(i+1)=sum(w.^i.*Q); U(i+2)=sum(w.^(i+1).*(P.^2+Q.^2));
end
ii=[1, 1]; for i=1:nA, ii=[ii -ii(2*i-1:2*i)]; end
Gxx=L(1:nA).*ii(1:nA); Gxy=S(2:nB+2).*ii(1:nB+1);
Gyy=U(3:nB+3).*ii(1:nB+1);
for i=1:nA-1
   Gxx=[Gxx; -Gxx(i,2:nA) L(nA+i)*ii(nA-i)];
   Gxy=[Gxy; -Gxy(i,2:nB+1) S(nB+2+i)*ii(nA-i)];
end
for i=1:nB, Gyy=[Gyy; -Gyy(i,2:nB+1) U(nB+i+3)*ii(nB+1-i)]; end
V=[S(1:nA).*ii(1:nA), U(2:nB+2).*ii(1:nB+1)]'; G=[Gxx, Gxy; Gxy', Gyy];
V1=inv(G)*V; num=V1(nB+1:-1:1)'; den=[V1(nA+nB+1:-1:nB+2)' 1];
num=num/den(1); den=den/den(1);
```

例 4.21　假设在下面的频率范围 w 上已经测出了一个系统的频率响应数据值为

```
>> w=logspace(-1,1);
>> H = [0.9892-0.1073*i    0.9870-0.1176*i    0.9843-0.1289*i   -0.9812-0.1412*i,...
        0.9773-0.1545*i    0.9728-0.1691*i   -0.9673-0.1848*i    0.9608-0.2017*i,...
        0.9530-0.2200*i    0.9437-0.2396*i    0.9328-0.2605*i    0.9198-0.2826*i,...
        0.9047-0.3058*i    0.8869-0.3301*i    0.8662-0.3551*i    0.8424-0.3805*i,...
        0.8150-0.4060*i    0.7840-0.4310*i    0.7491-0.4549*i    0.7103-0.4771*i,...
        0.6677-0.4968*i    0.6216-0.5133*i    0.5725-0.5258*i    0.5210-0.5335*i,...
        0.4680-0.5361*i    0.4144-0.5331*i    0.3613-0.5242*i    0.3099-0.5098*i,...
        0.2613-0.4900*i    0.2164-0.4654*i    0.1762-0.4370*i    0.1413-0.4057*i,...
        0.1121-0.3728*i    0.0886-0.3393*i    0.0706-0.3064*i    0.0577-0.2753*i,...
        0.0489-0.2466*i    0.0436-0.2210*i    0.0406-0.1987*i    0.0391-0.1796*i,...
        0.0383-0.1635*i    0.0377-0.1499*i    0.0369-0.1385*i    0.0356-0.1287*i,...
        0.0339-0.1201*i    0.0318-0.1123*i    0.0293-0.1051*i    0.0266-0.0983*i,...
        0.0239-0.0919*i    0.0212-0.0857*i];
```

则可以通过 freq2tf() 函数辨识出系统的传递函数模型

```
>> [n,d]=freq2tf(w,H,3,4)
n =   1.0000    7.0000   24.0000   24.0000
d =   1.0000   10.0000   35.0000   50.0000   24.0000
```

事实上，前面给出的频率响应数据正是由传递函数模型

$$G(s)=\frac{24+24s+7s^2+s^3}{24+50s+35s^2+10s^3+s^4}$$

在频率范围 0.1 到 10 上计算得出的

```
>> num=[1, 7, 24, 24]; den=[1, 10, 35, 50, 24];
>> H=freqs(num,den,w);
```

这里调用 MATLAB 提供的 freqs() 函数可以从一个已知的线性模型和给定的频率范围求出系统的频率响应数据，freqs() 函数的调用格式为

```
H = freqs(B, A, W)
```

其中 W 为给定的频率范围向量，而 B 和 A 分别为线性系统的传递函数分子和分母的系数向量。

在 MATLAB 的信号处理工具箱中，依照前面的算法给出了一个辨识系统传递函数模型的 MATLAB 函数 invfreqs()，该函数的调用格式为

```
[B, A] = invfreqs(H, W, n, m)
```

其中 W 为由离散频率点构成的向量，n 和 m 分别为待辨识系统的分子和分母阶次，H 为复数向量，其实部和虚部为辨识时用到的实部和虚部数据。返回的 B 和 A 分别为辨识出传递函数分子和分母的系数向量，即系统的传递函数模型。如果给出系统的幅频 mag 和相频响应数据 phase，则可以由下面的方式来调用 invfreqs() 函数



```
[B, A] = invfreqs(mag.*exp(sqrt(-1)*phase), W, n, m)
```

该函数是 freqs() 函数的逆变换。

重新考虑前面例子中的模型，调用 freqs() 函数则可以容易地由该模型可以求出系统的频率响应数据，由此数据容易地辨识出系统的传递函数模型

```
>>  [num, den] = invfreqs(H, w, 3, 4)
num =  1.0000    7.0000   24.0000   24.0000
den =  1.0000   10.0000   35.0000   50.0000   24.0000
>>  norm(num-[1, 7, 24, 24])
ans =  5.4104e-007
>>  norm(den-[1, 10, 35, 50, 24])
ans =  9.2525e-007
```

从上面的分析可以看出，采用 invfreqs() 基本上可以辨识出系统的传递函数模型，但对 MATLAB 的标准来说，这样辨识的精度并不是很高。

除了由频率响应数据辨识原系统模型以外，还可以根据阶跃响应及脉冲响应数据对系统的传递函数进行辨识，其具体的想法是，首先由阶跃响应数据或脉冲响应数据获得相应的频率响应数据，然后再根据上面的方法来辨识原系统的模型。由脉冲响应数据求取频率响应数据的方法是很显然的，因为脉冲响应函数 $g(t)$ 和频率响应函数 $G(j\omega)$ 满足下面的关系

$$G(j\omega) = \int_0^{\infty} g(t)\mathrm{e}^{-j\omega t}\mathrm{d}t \simeq \int_0^{T_f} g(t)\mathrm{e}^{-j\omega t}\mathrm{d}t \tag{4.8.10}$$

其中 T_f 取得足够大就可以由数值积分的算法得出频率响应数据。这样由脉冲响应数据的辨识问题就转换成由频率响应辨识的问题了。另外，由控制理论可知，若已知系统的阶跃响应数据，则可以通过数值微分的方法得出系统的脉冲响应数据，从而可以最终由已知的方法辨识出系统的传递函数模型。

4.8.2 离散时间系统的最小二乘辨识方法

如果控制系统是由式 (4.1.4) 给出的离散时间差分方程模型，且测出系统的时间响应序列为 $\{u_i, y_i\}$ $(i = 1, 2, \cdots, M)$，其中 u_i 和 y_i 分别为系统在第 iT 采样时刻的实测输入和输出值，则最简单地可以通过著名的最小二乘算法来辨识出系统的模型。

假设对式 (4.1.4) 中给出的差分方程可以进行首一化，则该方程可以改写成

$$y_{i+n} = -g_2 y_{i+n-1} - \cdots - g_n y_{i+1} - g_{n+1} y_i + f_1 u_{i+m+1} + f_2 u_{i+m} + \cdots + f_{m+1} u_i \tag{4.8.11}$$



对测出的各组输入和响应数据来说，可以容易地列写出下面的矩阵表示形式

$$Y_M = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} -y_n & \cdots & -y_1 & u_m & \cdots & u_1 \\ -y_{n+1} & \cdots & -y_2 & u_{m+1} & \cdots & u_2 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ -y_{M-1} & \cdots & -y_{M-n} & u_{M-n+m} & \cdots & u_{M-n} \end{bmatrix} \begin{bmatrix} -g_2 \\ \vdots \\ -g_{n+1} \\ f_1 \\ \vdots \\ f_{m+1} \end{bmatrix} = X_M\theta \tag{4.8.12}$$

由上式显然可以看出 Y_M, X_M 及 θ 的构成方式。

当然这样得出的系数矩阵一般情况下有 $M > n$，所以该方程为超定方程，如果原系数矩阵的各行是线性无关的，则方程的解不存在，这样只能按某种准则来求出方程其它形式的解，如最小二乘解

$$\theta = (X_M^T X_M)^{-1} X_M^T Y_M \tag{4.8.13}$$

假设有列向量输入数据 u 和输出数据 y, 并指定了系统的分子和分母阶次 m, n ，则上面的算法可以很容易由 MATLAB 来实现，

```
function [num,den]=lsqident(u,y,m,n)
M=length(u);  Y=y(n+1:M);
for i=1:length(Y)
   X(i,:)=[-y(n+i-1:-1:i)' u(m+i:-1:i)'];
end
t=X\Y; den=[1 t(1:n)'];  num=t(n+1:length(t))';
```

我们知道， X\Y 这样的简单命令相当于 $(X^TX)^{-1}X^TY$ 给出的最小二乘运算。可见最小二乘系统辨识方法可以由上面几条 MATLAB 语句相当容易地设计出来。

例 4.22　假设已知离散时间控制系统的输入输出数据为

i	u_i	y_i	i	u_i	y_i
1	0.5297	0	9	0.9304	45.3881
2	0.6711	12.7905	10	0.8462	58.5563
3	0.0077	26.5880	11	0.5269	67.0288
4	0.3834	21.7074	12	0.0920	65.9618
5	0.0668	26.6961	13	0.6539	54.3274
6	0.4175	23.0073	14	0.4160	58.2037
7	0.6868	28.3666	15	0.7012	55.4204
8	0.5890	39.1277			

调用前面编制的 lsqident() 函数则可以得出 4 阶模型的辨识结果

```
>> [nn,dd]=lsqident(u,y,3,4)
nn =  24.1467  -67.7944   63.4768  -19.8209
dd =   1.0000   -3.6193    4.9124   -2.9633    0.6703
```

事实上，上面给出的数据是根据脉冲传递函数

$$G(z)=\frac{24.1467z^3-67.7944z^2+63.4768z-19.8209}{z^4-3.6193z^3+4.9124z^2-2.9633z+0.6703}$$

在一组随机数据的激励下仿真而得出的

```
>> num=[24.1467, -67.7944, 63.4768, -19.8209];
>> den=[1, -3.6193, 4.9124, -2.9633, 0.6703];
>> u=rand(15,1);
>> y=dlsim(num,den,u);
```

分析辨识的结果可见

```
>> norm(nn-num)
ans = 3.2647e-007
>> norm(dd-den)
ans = 2.2382e-008
```

虽然这样的误差对 MATLAB 来说是过于大了，但在一般应用下该误差是可以被接受的。

当然，作为一个独立的学科分支，系统辨识远不止限于解决这样简单的问题，除了这样的称为自回归 (auto regressive, 简称 AR) 模型的传递函数之外，系统辨识还将涉及有噪声干扰情况下的辨识问题，而且辨识的算法也不仅仅限于简单的最小二乘法，相应的辨识方法还有辅助变量法、广义最小二乘法等，即使对简单的 AR 问题，辨识的方法还可以有 Yule-Walker 法、Burg 法等。瑞典学者 Leonard Ljung 教授主持编写的 MATLAB 系统辨识工具箱实现了常用的大多数有效的系统辨识方法并配有很实用的阶次自动识别功能，调用起来也是很方便的。由于篇幅所限，在这里就不能详细介绍系统辨识的方法及工具箱的基本功能了，有兴趣的读者可以参阅文献 [7, 8] 或系统辨识工具箱的联机帮助与实例。

4.8.3 控制系统的模型降阶实例

在控制系统的研究中，模型降阶技术起着很重要的作用。模型降阶的目的就是使高阶系统由一个相对低阶的模型作尽可能好的近似，使得高阶模型可以依照对低阶的设计方法进行近似设计。

模型降阶技术首先由 Edward Davison 提出，其想法是降低原始系统的系数矩阵的阶次，并保留原来系统的主导特征值与一些重要的状态变量。除了这种方法以外还出现了多种多样的其它基于状态方程降阶算法，如聚类分析法、奇异摄动法、均衡实现法及最优 Hankel 范数近似法等。在标准的控制系统工具箱中提供了基于均衡实现的降阶算法程序，由该程序可以容易地求出指定阶次的降阶模型。



假设在均衡实现中可控及可观测的 Gram 矩阵均为 Σ, 则它可以被人为地分割成两个部分：亦即 $\Sigma = \mathrm{diag}(\Sigma_1, \Sigma_2)$，其中子矩阵 Σ_1 包含原来 A 矩阵的较大奇异值，而 Σ_2 包含 A 矩阵的较小奇异值，相应地可以将原始系统作如下的分割

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} A_{b,11} & A_{b,12} \\ A_{b,21} & A_{b,22} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} B_{b,1} \\ B_{b,2} \end{bmatrix} u, \quad C_b = \begin{bmatrix} C_{b,1} & C_{b,2} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{4.8.14}$$

这样如果截取掉对应于小奇异值的子系统，则降阶模型可以写成

$$\dot{z}_1 = A_{b,11} z_1 + B_{b,1} u, \quad \hat{y} = C_{b,1} z_1 + Du \tag{4.8.15}$$

显而易见，这样得出的降阶模型无法保证原系统的稳态值，如果想保持原系统稳态值，则可以将降阶后的模型写成

$$\begin{cases} \dot{x}_1 = \left(A_{b,11} - A_{b,12} A_{b,22}^{-1} A_{b,21}\right) x_1 + \left(B_{b,1} - A_{b,12} A_{b,22}^{-1} B_{b,2}\right) u \\ y = \left(C_{b,1} - C_{b,2} A_{b,22}^{-1} A_{b,21}\right) x_1 + \left(D - C_{b,2} A_{b,22}^{-1} B_{b,2}\right) u \end{cases} \tag{4.8.16}$$

值得指出的是，虽然这样的方法可以保持原系统的稳态值，但由于降阶系统的前馈传输矩阵为 $D - C_{b,2} A_{b,22}^{-1} B_{b,2}$，而一般情况下这一矩阵和原系统的 D 矩阵是不同的，这样降阶系统的初始响应值可能和原系统的不一致。

例 4.23 考虑下面的原系统模型

$$\dot{x} = \begin{bmatrix} -3 & 1 & 0 & -1 \\ -0.5 & -1 & 1 & -1 \\ -1.5 & 1 & -2 & 0 \\ -1.5 & 2 & 1 & -4 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} u, \quad y = [1 \quad 0 \quad -1 \quad 0] x$$

由前面介绍的均衡实现方法，可以很容易地求出系统的均衡实现

```
>> a=[-3,1,0,-1; -0.5,-1,1,-1; -1.5,1,-2,0; -1.5,2,1,-4];
>> b=[1; 0; 0; 0]; c=[1,0,-1,0];
>> [Ab,Bb,Cb,G]=balreal(a,b,c)
Ab = -1.2829    0.4033    0.1900   -0.0359
      0.4033   -1.7748   -1.5444    0.3144
      0.1900   -1.5444   -5.9201    2.8156
      0.0359   -0.3144   -2.8156   -1.0223
Bb = -0.9849
      0.1577
      0.0730
      0.0138
Cb = -0.9849    0.1577    0.0730   -0.0138
G =   0.3781    0.0070    0.0005    0.0001
```

由得出的 G 向量可以看出，主要有系统的前两个状态变量起作用，所以可以截取掉后两个状态变量，这样就能得出系统的降阶模型为

```
>>  A1=Ab(1:2, 1:2),  B1=Bb(1:2,:),  C1=Cb(:,1:2)
A1 = -1.2829    0.4033
      0.4033   -1.7748
B1 = -0.9849
      0.1577
C1 = -0.9849    0.1577
```

如果采用后一种降阶方法，则可以调用 MATLAB 控制系统工具箱中的 modred() 函数，该函数的调用格式为

```
[A2,B2,C2,D2]=modred(Ab,Bb,Cb,D,elim)
```

其中 Ab，Bb，Cb，D 为均衡实现系统的状态方程模型参数，elim 为要消去的状态变量序号，当然为获得较好的近似，消去的状态变量应该选为对应于 Gram 矩阵元素的数值较小的变量，返回的 A2，B2，C2，D2 为所获得的降阶模型。考虑这一实际问题，可见第 3 和第 4 状态变量对应的 Gram 矩阵元素较小，所以可以将 elim 变量设置为 [3,4]。这时可以由下面的 MATLAB 命令来求出降阶模型

```
>>  [A2,B2,C2,D2] = modred(Ab,Bb,Cb,0,[3,4]')
A2 = -1.2780    0.3634
      0.3634   -1.4466
B2 = -0.9830
      0.1424
C2 = -0.9830    0.1424
D2 = 7.1467e-004
```

与基于状态方程的模型降阶方法并行发展的基于传递函数的降阶方法也有很多引人瞩目的成果，基于传递函数的模型降阶方法又称作模型化简方法 [1]，早期出现的连分式降阶算法，Padé 近似方法、主导模态方法与基于 Routh 稳定性的降阶方法在模型降阶方面是很有其代表意义的，近年来出现的各种频率响应拟合方法与最优降阶方法等在实际中的应用也是越来越广，作者曾经编写了一个应用于控制系统降阶的 MATLAB 软件系统 [14] 来实现各种常用的降阶算法，并比较现有降阶方法的精度及探索适于范围。有关基于传递函数的各种模型降阶算法详细情况及适用范围请参见文献 [13]，在这里只以典型的 Padé 降阶算法为例介绍模型降阶算法的 MATLAB 实现，假设原系统的传递函数模型为

$$G(s)=\frac{b_1s^{n-1}+b_2s^{n-2}+\cdots+b_{n-1}s+b_n}{s^n+a_1s^{n-1}+\cdots+a_{n-1}s+a_n} \tag{4.8.17}$$

则可以将其展开成

$$G(s)=\sum_{j=1}^{\infty}c_js^{j-1} \tag{4.8.18}$$

式中 $c_1=b_n/a_n$，且对 $i=2,3,\cdots$ 存在

$$c_i=\frac{1}{a_n}\Big(b_{n+1-i}-\sum_{j=1}^{i-1}a_{n-j}c_{i-j}\Big) \tag{4.8.19}$$



这样可以写出一个 m 阶降阶模型 $G_{r/m}(s)$

$$G_m^r(s) = \frac{\beta_{r+1}s^r + \beta_r s^{r-1} + \cdots + \beta_1}{\alpha_{m+1}s^m + \alpha_m s^{m-1} + \ldots + \alpha_1} \tag{4.8.20}$$

其中 $\alpha_1 = 1$, $\beta_1 = c_1$, 且 α_i, $i = 2, \ldots, m+1$ 和 β_i, $i = 2, \ldots, k+1$ 系数可以由下面的式子容易地求出

$$Wx = w, \quad v = Vy \tag{4.8.21}$$

其中

$$\begin{aligned} x &= [\alpha_2, \alpha_3, \cdots, \alpha_{m+1}]^T, \quad w = [-c_{r+2}, -c_{r+3}, \cdots, -c_{m+r+1}]^T \\ v &= [\beta_2 - c_2, \beta_3 - c_3, \cdots, \beta_{r+1} - c_{r+1}]^T, \quad y = [\alpha_2, \alpha_3, \cdots, \alpha_{r+1}]^T \end{aligned} \tag{4.8.22}$$

且

$$W = \begin{bmatrix} c_{r+1} & c_k & \cdots & 0 & \cdots & 0 \\ c_{r+2} & c_{r+1} & \cdots & c_1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \ddots & \vdots \\ c_{r+m} & c_{r+m-1} & \cdots & c_{m-1} & \cdots & c_{r+1} \end{bmatrix} \tag{4.8.23}$$

$$V = \begin{bmatrix} c_1 & 0 & 0 & \cdots & 0 \\ c_2 & c_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_r & c_{r-1} & c_{r-2} & \cdots & c_1 \end{bmatrix} \tag{4.8.24}$$

从上面的算法中可以看出，这里涉及了大量的矩阵运算，当然在掌握了 MATLAB 这一方便的语言之后，问题就容易解决了。用户可以用 MATLAB 语言编写出下面程序段

```
function [nred,dred]=padered(num,den,r,m)
n=length(den)-1; k=length(num)-1;
d0=den(n+1); L=m+r;
num0=[zeros(1,L+n-k-1) num]/d0; den0=[zeros(1,L) den]/d0;
c=zeros(1,m+r); c(1)=num0(n+L);
for i=2:m+r+1
   c(i)=num0(n+1+L-i)-sum(den0(n-i+2+L:n+L).*c(1:i-1));
end
w=-c(r+2:m+r+1)';
vv=[c(r+1:-1:1)'; zeros(m-1-r,1)];
W=rot90(hankel(c(m+r:-1:r+1),vv));
V=rot90(hankel(c(r:-1:1)));
x=[1 (W\w)']; dred=x(m+1:-1:1)/x(m+1);
y=[1 x(2:r+1)*V'+c(2:r+1)]; nred=y(r+1:-1:1)/x(m+1);
```




由上面的程序段可以生成一个 MATLAB 函数文件 pademod.m，该函数的输入参数的含义分别为：num，den 为原系统的分子和分母多项式的系数，r，m 分别表示降阶后模型的分子分母阶次，而 nred，dred 分别返回降阶后模型的分子和分母多项式系数。如果用户想获得一个 Padé 降阶模型，则可以直接调用此函数来完成。

例 4.24 假设原始系统的传递函数模型为

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

调用 pademod() 函数来求二阶降阶模型，则可以得出

```
>> num=[1,7,24,24]; den=[1,10,35,50,24];
>> [nred,dred]=pademod(num,den,1,2)
nred =    0.7304    2.5043
dred =    1.0000    3.4435    2.5043
```

其中可以查出降阶过程的中间结果为

```
c =    1.0000   -1.0833    1.0903   -1.0666
W =   -1.0833    1.0000
       1.0903   -1.0833
V =    1.0000
```

习　题

1) 将下列系统的状态方程模型用 MATLAB 表示出来

(a) $\dot{x}=\begin{bmatrix}5&2&1&0\\0&4&6&0\\0&-3&-5&0\\0&-3&-6&-1\end{bmatrix}x+\begin{bmatrix}1\\2\\3\\4\end{bmatrix}u,\ y=[1,2,5,2]x$

(b) $\dot{x}=\begin{bmatrix}2&2&1\\1&3&1\\1&2&2\end{bmatrix}x+\begin{bmatrix}3\\3\\4\end{bmatrix}u,\quad y=x_1$

2) 将下面的传递函数模型用 MATLAB 表示出来

(a) $G_1(s)=\dfrac{s^4+35s^3+291s^2+1093s+1700}{s^9+9s^8+66s^7+294s^6+1029s^5+2541s^4+4684s^3+5856s^2+4629s+1700}$

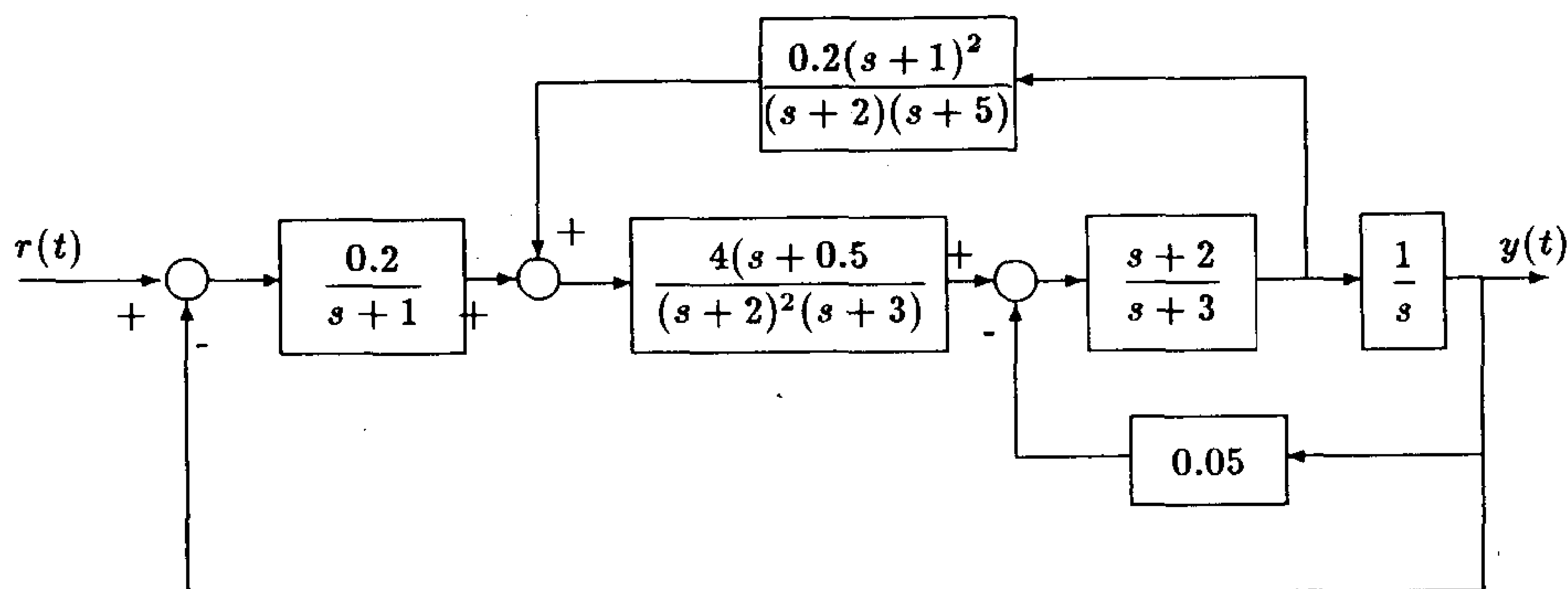
(b) $G_2(s)=\dfrac{15(s+3)}{(s+1)(s+5)(s+15)}$

(c) $G_3(s)=\dfrac{100s(s+2)^3(s^2+3s+2)^2}{(s+1)(s-1)(s^3+3s^2+5s+2)((s+1)^2+3)^2}$

(d) $G_4(z)=\dfrac{24.1467z^3-67.7944z^2+63.4768z-19.8209}{z^4-3.6193z^3+4.9124z^2-2.9633z+0.6703}$

3) 求出 1) 与 2) 题各个系统的零极点，分析其稳定性，并判断它们是否为最小相位系统。

4) 求出下图中所示系统的状态方程模型。



5) 求出习题 2) 中各个系统模型的等效状态方程。

6) 求出习题 1) 中各个习题的等效传递函数模型。

7) 给定时间延迟系统模型 $G(s)=\mathrm{e}^{-5s}/(s+1)$，试在采样周期 $T=0.1$ 下将其离散化成状态方程模型，并说明得出的模型特性。

8) 已知连续系统的状态方程为

$$\dot{x}=\begin{bmatrix}0&1&1&1\\0&0&0&1\\0&1&0&0\\0&0&1&1\end{bmatrix}x+\begin{bmatrix}1&0\\0&0\\0&1\\1&0\end{bmatrix}u$$

试在采样周期为 $T=0.01$ 和 $T=1$ 时对之进行离散化处理，并在得出离散化状态方程之后再将之变换成连续的模型，比较两个采样周期下变换结果的精度，并得出初步结论。

9) 仿照 c2dm() 和 d2cm() 及给出的传递函数离散化、连续化函数编写出名为 tfc2dm() 和 tfd2cm() 的转换函数，并对实际例子采用不同的方法进行离散化和连续化来验证这两个函数。

10) 若不调用控制系统工具箱的 c2d() 函数，而想从定义直接实现离散化功能，试由 MATLAB 编写出相应的函数。(提示：$F=\mathrm{e}^{AT}$ 当然可以由指数函数直接求出，而求 G 公式中的积分可以通过幂级数的方式来求得。)

11) 求出习题 1) 中各个模型的可控、可观测及 Jordan 标准型。

12) 求出习题 2) 中模型的可控标准型实现，并由它得出系统的可观测及 Jordan 标准型。

13) 求出下面多变量系统模型的可控或可观测标准型实现

$$\text{(a)}.\ A=\begin{bmatrix}0&1&1&1\\0&0&0&1\\0&1&0&0\\0&0&1&1\end{bmatrix},\ B=\begin{bmatrix}1&0\\0&0\\0&1\\1&0\end{bmatrix}\quad \text{(b)}.\ A=\begin{bmatrix}0&2&0&0\\0&1&-2&0\\0&0&3&1\\1&0&0&0\end{bmatrix},\ C=\begin{bmatrix}0&1&0&0\\0&0&1&0\end{bmatrix}$$

14) 求出下面状态方程模型的最小实现

$$\dot{x}=\begin{bmatrix}0&-3&0&0\\1&-4&0&0\\0&0&0&0\\0&0&1&-2\end{bmatrix}x+\begin{bmatrix}3&2\\1&2\\1&1\\1&1\end{bmatrix}u,\quad y=\begin{bmatrix}0&1&0&0\\0&0&0&1\end{bmatrix}x$$

15) 依照前面给出的算法用 MATLAB 编写出对状态方程进行 Kalman 分解的函数，使其调用格式为

```
[Ak, Bk, Ck, Dk, Tk]=kalmdcp(A, B, C, D, key)
```

其中原系统模型为 (A,B,C,D)，变量 key 为开关量，使得它等于 1 时作出 Kalman 分解模型 (Ak,Bk,Ck,Dk)，而等于其它值时求出最小实现模型 (Ak,Bk,Ck,Dk)，其中 Tk 为变换矩阵。用编写的程序段对前面的问题进行最小实现处理，并和前面得出的结果进行比较。

16) 假设一个系统的频率响应实测数据为

ω_i	P_i	Q_i	ω_i	P_i	Q_i	ω_i	P_i	Q_i
0.1000	1.0048	0.0197	0.4942	1.1139	0.0683	2.4421	0.5802	-1.6284
0.1099	1.0058	0.0216	0.5429	1.1367	0.0680	2.6827	0.2393	-1.5091
0.1207	1.0070	0.0237	0.5964	1.1637	0.0654	2.9471	-0.0040	-1.3105
0.1326	1.0084	0.0259	0.6551	1.1957	0.0594	3.2375	-0.1491	-1.0931
0.1456	1.0102	0.0283	0.7197	1.2335	0.0488	3.5565	-0.2203	-0.8933
0.1600	1.0122	0.0310	0.7906	1.2778	0.0315	3.9069	-0.2443	-0.7247
0.1758	1.0148	0.0338	0.8685	1.3292	0.0050	4.2919	-0.2412	-0.5882
0.1931	1.0178	0.0368	0.9541	1.3879	-0.0342	4.7149	-0.2243	-0.4799
0.2121	1.0215	0.0400	1.0481	1.4537	-0.0909	5.1795	-0.2014	-0.3944
0.2330	1.0259	0.0434	1.1514	1.5244	-0.1713	5.6899	-0.1768	-0.3270
0.2560	1.0312	0.0470	1.2649	1.5956	-0.2837	6.2506	-0.1529	-0.2736
0.2812	1.0376	0.0506	1.3895	1.6574	-0.4373	6.8665	-0.1308	-0.2310
0.3089	1.0453	0.0543	1.5264	1.6919	-0.6403	7.5431	-0.1112	-0.1967
0.3393	1.0545	0.0580	1.6768	1.6702	-0.8935	8.2864	-0.0939	-0.1689
0.3728	1.0656	0.0615	1.8421	1.5546	-1.1775	9.1030	-0.0790	-0.1462
0.4095	1.0789	0.0645	2.0236	1.3164	-1.4410	10.0000	-0.0663	-0.1273
0.4498	1.0949	0.0669	2.2230	0.9684	-1.6091			

试根据本章中给出的方法辨识出系统的传递函数模型。(提示：原系统模型为 3 阶模型)

17) 已知一个离散时间系统的输入输出数据为

i	u_i	y_i	i	u_i	y_i	i	u_i	y_i
1	0.9103	0	9	0.9910	54.5252	17	0.6316	62.1589
2	0.7622	18.4984	10	0.3653	65.9972	18	0.8847	63.0000
3	0.2625	31.4285	11	0.2470	62.9181	19	0.2727	68.6356
4	0.0475	32.3228	12	0.9826	57.5592	20	0.4364	60.8267
5	0.7361	28.5690	13	0.7227	67.6080	21	0.7665	57.1745
6	0.3282	39.1704	14	0.7534	70.7397	22	0.4777	60.5321
7	0.6326	39.8825	15	0.6515	73.7718	23	0.2378	57.3803
8	0.7564	46.4963	16	0.0727	74.0165	24	0.2749	49.6011

用最小二乘法及 Yule-Walker 法辨识出系统的脉冲传递函数模型 (提示：原系统的阶次为 3，在辨识过程中可以调用 MATLAB 系统辨识工具箱中提供的 ar() 函数。)

18) 求出下面传递函数模型的均衡实现降阶模型及 Padé 降阶模型

(a) $G(s)=\dfrac{0.067s^5+0.6s^4+1.5s^3+2.016s^2+1.55s+0.6}{0.067s^6+0.7s^5+3s^4+6.67s^3+7.93s^2+4.63s+1}$，求 $G_{1/2}(s)$



(b) $G(s)=\dfrac{3s^4+2s^3+s^2+4s+2}{3s^5+5s^4+s^3+2s^2+2s+1}$，求 $G_{1/3}(s)$

并分析降阶模型及原系统的稳定性。

参 考 文 献

[1] Atherton D P, Borne P (eds.). Concise encyclopedia of modelling and simulation. Oxford: Pergamon Press, 1992

[2] Balasubramanian R. Continuous time controller design. IEE Control Engineering Series. Vol. 39. London: Peter Peregrinus Ltd, 1989

[3] Frederick D K, Rimer M. Benchmark problem for CACSD packages. Abstracts of the second IEEE symposium on computer-aided control system design. Santa Barbara. USA, 1985

[4] Furuta K, Sano A, Atherton D P. State variable methods in automatic control. London: John Wiley and Sons, 1988

[5] Kailath T. Linear systems. Englewood Cliffs: Prentice Hall, 1980

[6] Laub A J, Heath M T, Paige C C *et al.* Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms. IEEE Trans. Automatic Control, 1987, AC-32(1): 115-122.

[7] Ljung L. System identification – theory for the user. Englewood Cliffs: Prentice-Hall, 1987

[8] Ljung L. System identification toolbox – user's guide. MathWorks, 1988

[9] Maciejowski J M. Multivariable feedback design. Wokingham: Addison-Wesley, 1989

[10] Moore B. Principal component analysis in linear systems: controllability, observability, and model reduction. IEEE Trans. Automatic Control, 1981, AC-26(1): 17-32

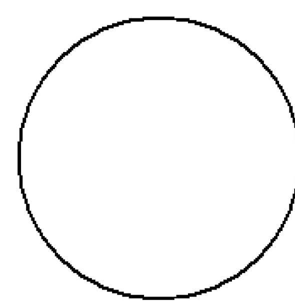
[11] Rimer M, Frederick D K. Solutions of the Grumman F-14 benchmark control problem. IEEE Control Systems Magazine, 1987: 36-40

[12] Rosenbrock H H. Computer aided control systems design. London: Academic Press, 1973

[13] Xue D. Model reduction approaches for control systems. Internal Lecture Notes. Shenyang: Northeastern University, 1993

[14] Xue D, Atherton D P. A menu-driven model reduction program and its applications. In: Barker H (ed.). Computer-aided design in control systems. Oxford: Pergamon Press, 1992. 232-238

[15] Xue D, Atherton D P. BLOCKM – a block diagram modelling interface and its applications. Proceedings IEEE Symposium on CACSD. Napa, California, USA, 1992. 242-249



第 5 章 控制系统的计算机辅助频域与时域分析

以往如果我们想获得一个控制系统的响应，往往需要自己去编写一个数值计算的程序，例如如果想求得系统的阶跃响应数据并绘制阶跃响应曲线，则首先应该编写一个求解微分方程的子程序，然后将原系统模型输入给计算机，通过计算机求出阶跃响应数据，然后再编写一个画图子程序，将所得的数据以曲线的方式绘制出来，一般情况下求解这样简单的问题则需要花费很多的时间，并且求出的结果连用户自己也不敢保证是正确的。MATLAB 这样的高性能软件及语言出现以来，特别是 MATLAB 的控制系统工具箱及 SIMULINK 辅助环境问世以来，确实给系统分析者带来了福音，因为用户不必像以前那样将时间和精力无谓地花费在一些无关紧要的工作中去。

在本节中将首先介绍控制系统的计算机辅助频率响应分析和曲线绘制，然后侧重控制系统计算机辅助分析，特别地，如何利用 MATLAB 下一个使用十分方便的软件 SIMULINK 来对给定系统进行仿真，用户将从中看出以往十分困难的系统仿真问题可以由 SIMULINK 轻而易举地解决。然后将介绍连续随机输入系统的仿真算法，在本章的最后还将介绍非线性系统的频域响应分析方法。

5.1 控制系统的频域响应

5.1.1 频率响应的计算方法

在控制系统实验中可以发现，若一个线性系统受到频率为 ω 的正弦信号激励时，它的输出仍然为正弦信号，但其幅值与输入信号成 $M(\omega)$ 比例关系，而且输出与输入信号之间有一个相位差 $\phi(\omega)$，这种比例和相位差随 ω 的变化而变化，这样就可以通过 $M(\omega)$ 和 $\phi(\omega)$ 来表示系统的特性了；这种表示方法即为系统的频域分析方法。

假设已知系统的传递函数模型为

$$G(s)=\frac{b_1s^m+b_2s^{m-1}+\cdots+b_ms+b_{m+1}}{a_1s^n+a_2s^{n+1}+\cdots+a_ns+a_{n+1}} \tag{5.1.1}$$

则该系统的频率响应数据可以由

$$G(j\omega)=\frac{b_1(j\omega)^m+b_2(j\omega)^{m-1}+\cdots+b_m(j\omega)+b_{m+1}}{a_1(j\omega)^n+a_2(j\omega)^{n+1}+\cdots+a_n(j\omega)+a_{n+1}} \tag{5.1.2}$$

直接求出。具体方法可以由下面的语句来实现，如果有一个频率向量 w，则

```
Gw=polyval(num, sqrt(-1)*w)./polyval(den, sqrt(-1)*w);
```

其中 num 和 den 分别为系统的分子分母多项式系数向量。求取一个系统的频率响应数据



其实存在多种算法，上面的算法实现起来很简单，但有时效果并不是很理想[7]，因为它涉及到多项式的求值问题，而这样的问题在多项式阶次较高时会是相当困难的。若已知系统的状态方程模型，则最好能直接采用矩阵求逆的方法来求解频率响应，若一个系统的状态方程参数为 (A,B,C,D)，则可以由下面的公式直接求出频率响应数据

$$G(j\omega)=C(j\omega I-A)^{-1}B+D \tag{5.1.3}$$

为了减小运算量，则可以首先对 A 矩阵进行 Hessenberg 变换，即求出一个变换矩阵 T 使得 $H=T^TAT$，其中 H 矩阵为具有下面格式的 Hessenberg 矩阵

$$H=\begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1,n-1} & h_{1n} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2,n-1} & h_{2n} \\ 0 & h_{32} & h_{33} & \cdots & h_{3n} & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n,n-1} & h_{nn} \end{bmatrix} \tag{5.1.4}$$

亦即第 1 次主对角线下的所有元素均为 0 。Hessenberg 变换可以通过 MATLAB 函数 hess() 来直接求取，该函数的调用格式是很直观的

```
[T, H]=hess(A)
```

这时若定义 $\hat{B}=T^TB$, $\hat{C}=CT$, 则式 (5.1.3) 中的频率响应求值公式可以简化成

$$G(s)=\hat{C}(sI-H)^{-1}\hat{B}+D \tag{5.1.5}$$

其中 $(sI-H)^{-1}\hat{B}$ 可以调用 ltifr() 函数来直接求出，该函数的调用格式为

```
G=ltifr(H, B, sqrt(-1)*w)
```

式中 sqrt(-1)*w 是用来构造 $j\omega$ 向量的，这样由返回的复数矩阵 G 可以容易地求出原系统的频率响应数据来

```
fr=C*G+D*ones(q,length(w)))
```

其中 q 为输出变量的个数，此外，这里给出的 ltifr() 函数可以直接用来求取多变量系统的频率响应数据 fr，其中 ltifr() 函数是一个内部函数，其算法如果用 MATLAB 函数来表示则可以写成

```
function G=ltifr(A,B,s)
ns = length(s); na=length(A);
e = eye(na); G = sqrt(-1)*ones(na,ns);
for i=1:ns
   G(:,i) = (s(i)*e-A)\B;
end
```




5.1.2 频率响应曲线绘制

MATLAB 提供了多种求取并绘制系统频率响应曲线的函数，如 Bode 图绘制函数 bode()， Nyquist 曲线绘制函数 nyquist()[1)]，以及 Nichols 曲线绘制函数 nichols() 等，其中 bode() 函数的调用格式为

```
[m,p]=bode(num,den,w);
```
或
```
[m,p]=bode(A,B,C,D,iu,w);
```

这里 num，den 和 A，B，C，D 分别为系统的传递函数或状态方程的参数，而 w 为频率点构成的向量，该向量最好由 logspace() 函数来构成。和前面的叙述一样， iu 为一个数值，反映要求取的输入信号标号，当然对单输入系统来说， iu=1 。bode() 函数本身可以通过输入元素的个数来自动地识别给出的是传递函数模型还是状态方程模型，从而可以正确地求出 Bode 响应的幅值向量 m 与相位向量 p，有了这些数据之后就可以由下面的 MATLAB 命令

```
subplot(211); semilogx(w,20*log10(m))
subplot(212); semilogx(w,p)
```

在同一个窗口上同时绘制出系统的 Bode 响应曲线了，其中前面一条命令中对得出的 m 向量求取分贝 (dB) 值。如果用户只想绘制出系统的 Bode 图，而对获得幅值和相位的具体数值并不感兴趣，则可以由如下更简洁的格式调用 bode() 函数

```
bode(A, B, C, D, iu, w);
```
或
```
bode(num, den, w);
```

或更简洁地

```
bode(A, B, C, D, iu);
```
或
```
bode(num, den);
```

这时该函数会自动地根据模型的变化情况选择一个比较合适的频率范围。

值得注意的是，除了前面给出的最基本调用格式之外，根据实际需要的不同，求取系统的频率响应数据还有很多的变形方式，如求取系统的 Nyquist 响应数据的函数 nyquist()，求取 Nichols 特性数据的函数 nichols() 等，这些函数的调用格式和 bode() 一样也是十分直观的，例如 nichols() 函数的调用格式为

```
[mag,pha]=nichols(num,den,w);
```
或
```
[mag,pha]=nichols(A,B,C,D,iu,w);
```

其中该函数仍然返回系统的幅值响应数据 mag 和相位响应数据 pha，可见该函数的调用格式和 bode() 函数是完全一致的，得出的结果也是相同的，事实上虽然使用的算法不同，这两个函数得出的结果也是基本一致的。但 Nichols 图的绘制方式和 Bode 图是不同的，一个系统的 Nichols 图可以由下面的命令简单地绘制出来

[1)] 其中较通用的控制系统工具箱 3.0a, b 版本中 nyquist() 函数中存在错误，为保证其正确运行，用户可以简单地将该文件中首句中的返回变量名 im 改写成 imt 。



plot(phase, mag) 或 plot(phase,20*log10(mag))

当然用下面的格式调用 nichols() 可以直接绘制出系统的 Nichols 曲线

nichols(num, den, w); 或 nichols(A, B, C, D, iu, w);

MATLAB 还提供了更直接的求取频率响应数据的函数 freqresp()，其调用格式为

```
[x,y]=freqresp(num,den,sqrt(-1)*w); 或
[x,y]=freqresp(A,B,C,D,iu,sqrt(-1)*w);
```

注意，这里给出的频率向量为 sqrt(-1)*w，而不是 w。

在分析系统性能的时候经常涉及到系统的幅值与相位裕度的问题，使用控制系统工具箱提供的 margin() 函数可以直接求出系统的幅值与相位裕度，该函数的调用格式为

```
[Gm, Pm, Wcg, Wcp]=margin(A, B, C, D) 或
[Gm, Pm, Wcg, Wcp]=margin(num, den)
```

可以看出，该函数能直接由系统的状态方程或传递函数模型来求取系统的幅值裕度 Gm 和相位裕度 Pm，并求出幅值裕度和相位裕度处相应的频率值 Wcg 和 Wcp。除了由系统模型直接求取幅值和相位裕度之外，MATLAB 的控制系统工具箱中还提供了由幅值和相位响应数据来求取裕度的方法，这时函数的调用格式为

```
[Gm, Pm, Wcg, Wcp]=margin(MAG, PHASE, w)
```

其中 MAG 和 PHASE 可以是由 bode() 类函数获得的幅值和相位向量，也可以是系统的实测幅值与相位向量，w 为相应的频率点向量。

例 5.1 考虑一个线性系统模型

$$\dot{x}(t)=\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\-62.5&-213.8&-204.2&-54\end{bmatrix}x(t)+\begin{bmatrix}0\\0\\0\\1\end{bmatrix}u(t),\quad y(t)=[1562,1875,0,0]x(t)$$

试绘制出系统在 0.1 到 10 之间的频率范围上 Bode 图和 Nyquist 图。

要解决这样的问题，首先应该将系统模型与频率向量输入给计算机，然后调用相应的函数来绘制出相应的频率响应图形。可以给出下面的 MATLAB 命令

```
>> w=logspace(-1,1);
>> A=[0,1,0,0; 0,0,1,0; 0,0,0,1; -62.5,-213.8,-204.2,-54];
>> B=[0; 0; 0; 1]; C=[1562,1875,0,0]; D=0;
>> [m, p]=bode(A,B,C,D,1,w);
>> subplot(211), semilogx(w,20*log10(m))
>> subplot(212), semilogx(w,p)
>> [x,y]=nyquist(A,B,C,D,1,w);
>> clg; plot(x,y)
```

这组命令所得出的 Bode 图和 Nyquist 图分别如图 5-1 和 5-2 所示。

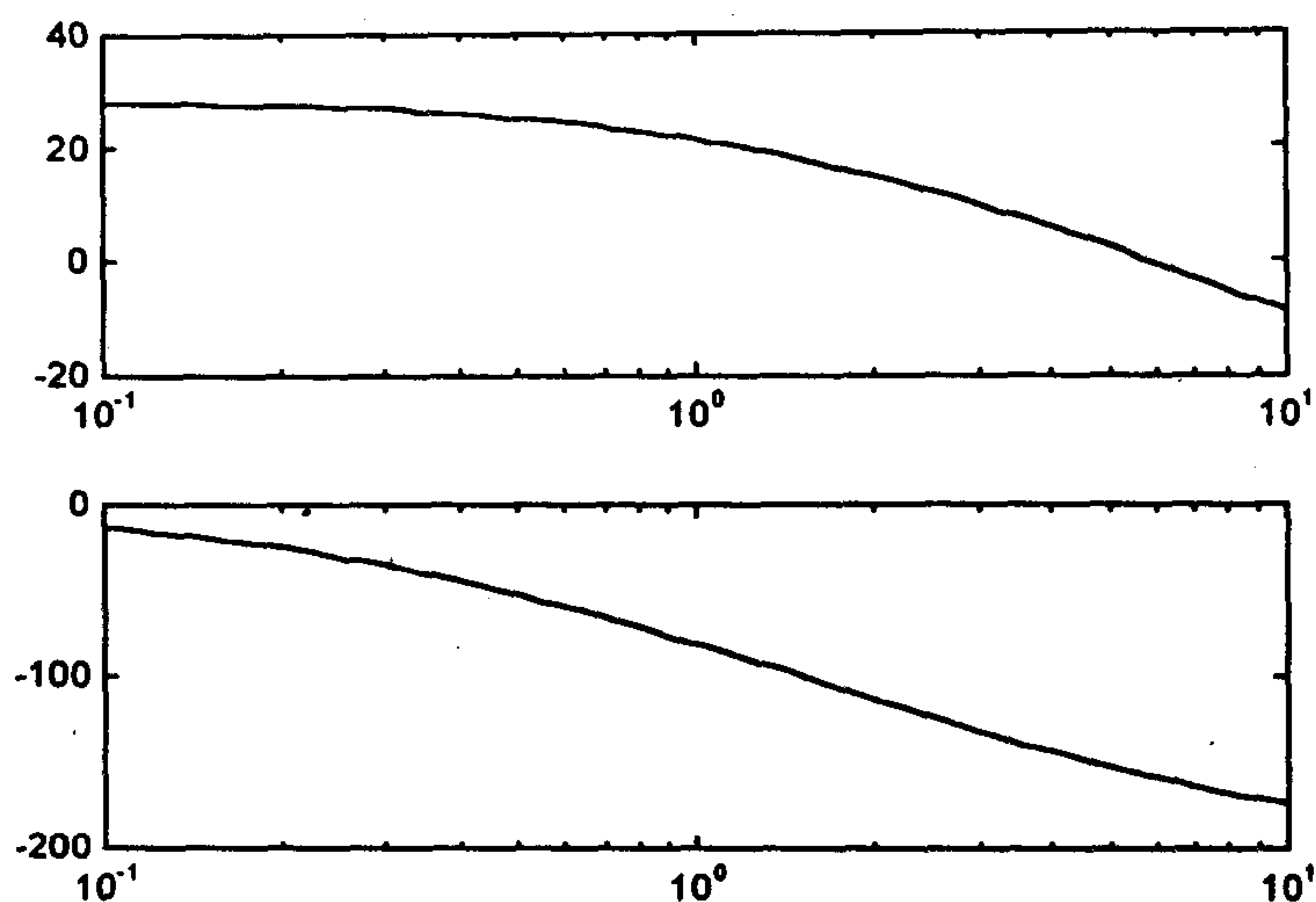


图5-1 控制系统的 Bode 曲线

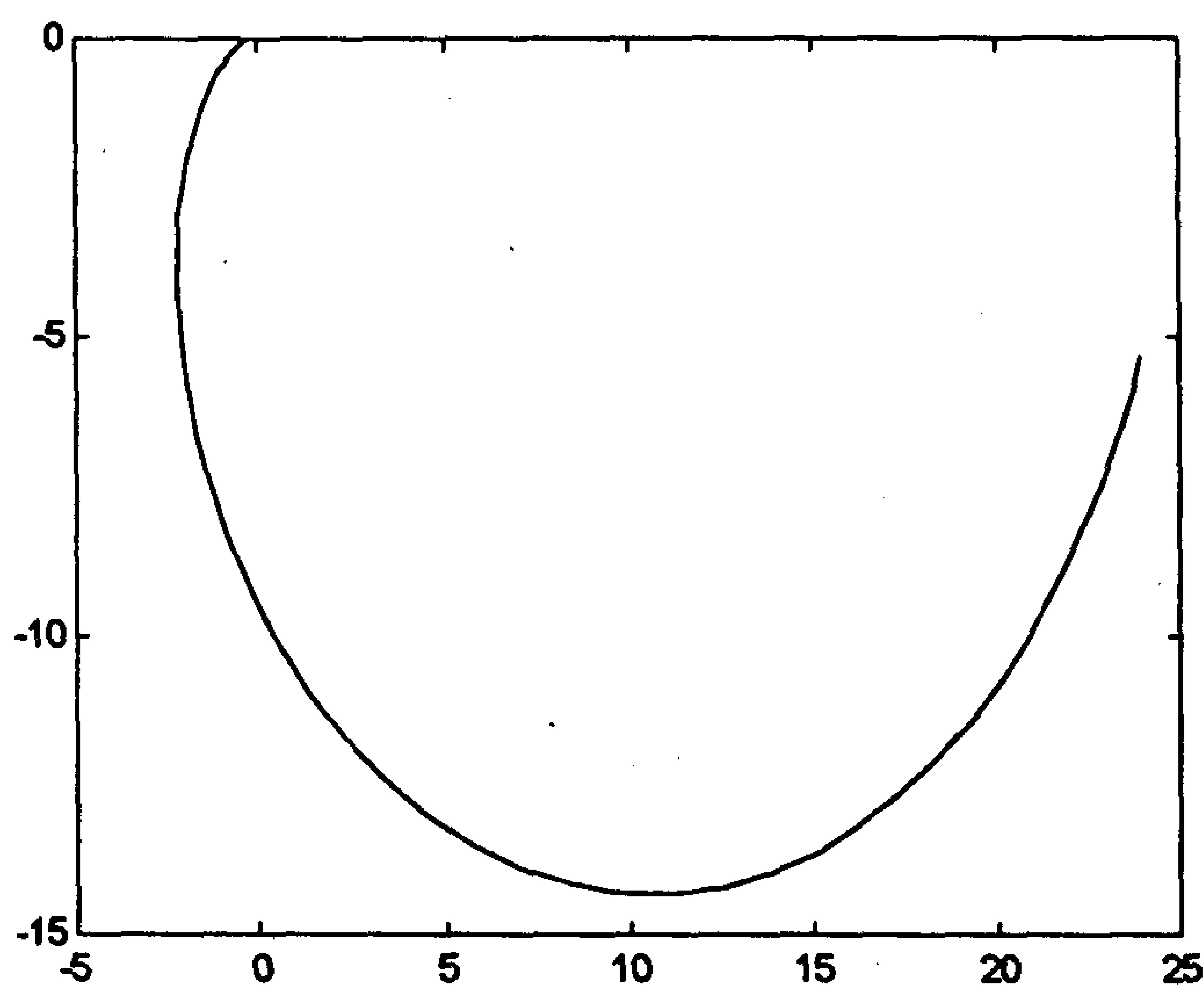


图5-2 控制系统的 Nyquist 曲线

例如对前面的例子来说，若给出 bode(A,B,C,D,1,w) 命令则可以得出如图 5-3 所示的 Bode 图形，可以看出和图 5-1 的唯一区别在于在图 5-3 中对原始图形做了一些修饰，如添加了网格线，并加入了横纵坐标的说明文字，使得 Bode 图形显得更规范化。

利用 MATLAB 控制系统工具箱提供的 margin() 函数可以容易地求出系统的幅值与相位裕度及其发生处的频率值



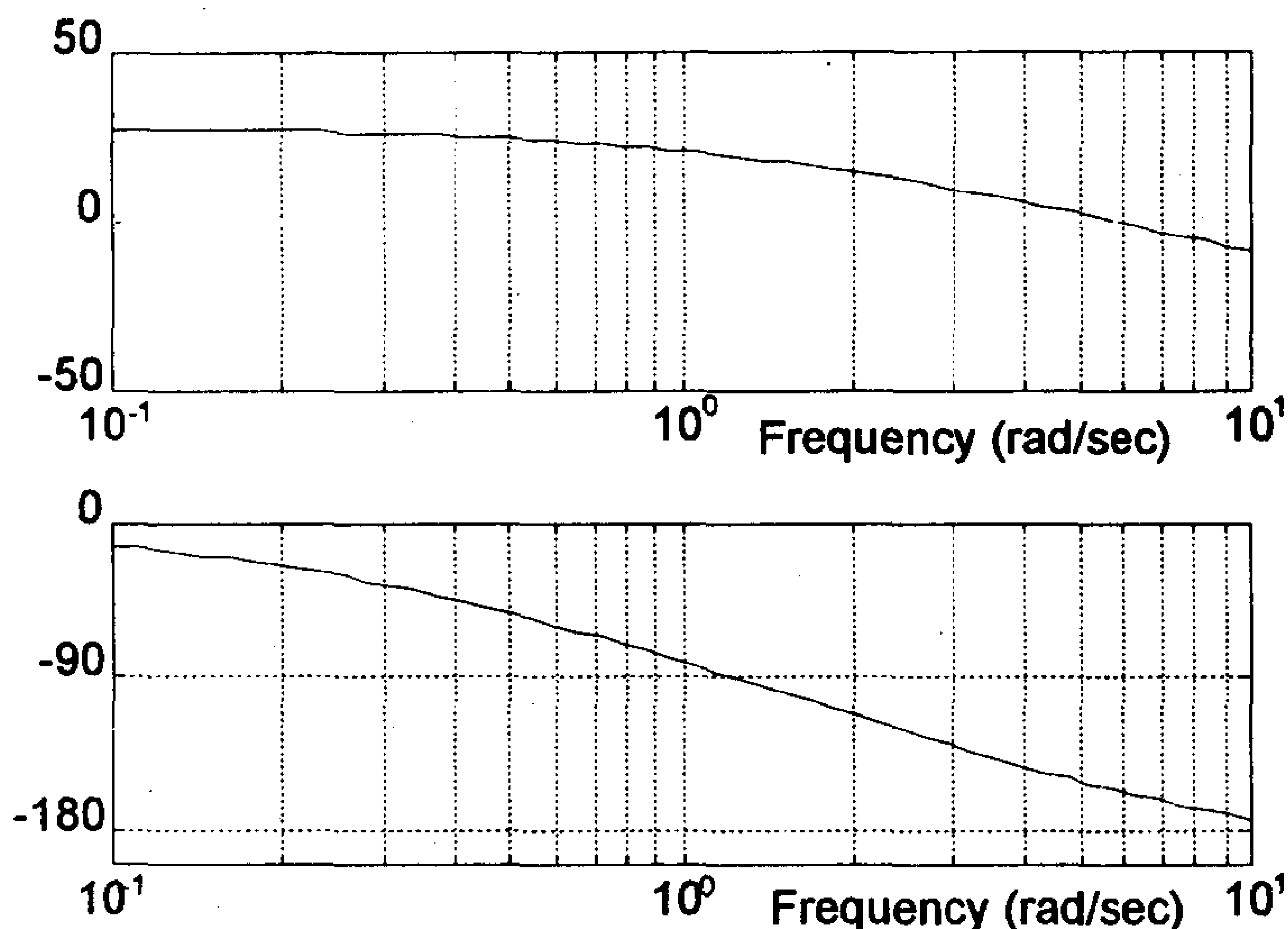


图5-3　控制系统的 Bode 曲线

```
>>  [Gm, Pm, Wcg, Wcp]=margin(A, B, C, D)
Gm =     4.4922
Pm =    23.0705
Wcg =   12.6466
Wcp =    5.8275
```

可见在 $\omega=12.6466$ 处相位穿越 180° 线，这时幅值裕度的值为 4.4922，而 $\omega=5.8275$ 时系统的幅值为 1，且相位裕度的值为 23.0705°。当系统不存在幅值或相位裕度时，幅值相位裕度的值应该返回 Inf，而发生的频率值应该返回 NaN。值得指出的是，当给出的是频率响应数据而不是系统的模型时，有时可能得不出系统的裕度，例如在上面的例子中 $\omega=12.6466$ 已经超出了给定的频率范围，故将得不出幅值裕度。

5.1.3　离散时间系统的频率响应分析

若离散时间系统的状态方程模型为 (F,G,C,D)，则该系统的频率响应数据可以由下面的公式得出.

$$G(j\omega)=C(\mathrm{e}^{j\omega}I-F)^{-1}G+D \tag{5.1.6}$$

这里 ω 为计算点的频率值。同样，对传递函数模型来说，只需将连续时的频率值 $j\omega$ 替换为 $\mathrm{e}^{j\omega}$ 即可以得出离散时间系统的频率响应数值。

离散时间系统的频率分析也可以调用相应的控制系统工具箱函数来完成，比如说离散时间系统的 Bode 图形可以由 dbode() 函数来求出，该函数在名称上比连续系统 Bode 图形绘制函数 bode() 多了一个 d 以示为离散时间系统，在控制系统工具箱中这样的函数命名方式是相当普遍的，例如离散时间系统的 Nyquist 图形数据可以相应地由 dnyquist() 函数来求出，而离散时间系统的 Lyapunov 函数可以由 dlyap() 函数来求

解，后面要介绍的离散时间系统的阶跃响应数据可以由 dstep() 函数来求取。dbode() 函数的调用格式为

`[mag,phase]=dbode(F,G,C,D,Ts,iu,w)` 或 `[mag,phase]=dbode(num,den,w);`

其中 (F,G,C,D) 为系统的离散时间状态方程模型的参数，Ts 为采样周期，iu 为输入序号，w 仍为频率向量。在后一种调用格式中，num 和 den 分别为离散时间系统传递函数模型的分子和分母多项式系数构成的向量，调用这样的函数返回的 mag 和 phase 则分别为该系统的幅值和相位向量，通过它们可以绘制出系统的频率响应曲线。当然这一函数的调用格式可以简化成下面的形式

`dbode(F,G,C,D,iu,w)` 或 `dbode(F,G,C,D,iu)`

在这种情况下，将直接绘制出离散时间系统的 Bode 图，而不是返回频率响应数据。

例 5.2　考虑下面的离散时间状态方程模型

$$x(k+1)=\begin{bmatrix}-1 & -2 & -2\\ 0 & -1 & 1\\ 1 & 0 & -1\end{bmatrix}x(k)+\begin{bmatrix}2\\0\\1\end{bmatrix}u(k),\quad y(k)=[1,\ 2,\ 0]x(k)$$

假设系统的采样周期为 $T=0.1$，键入系统的数学模型之后，就可以调用 dbode() 函数绘制出频率响应图形来，具体的 MATLAB 命令如下

```
>> F=[-1,-2,-3; 0,-1,1; 1,0,-1];
>> G=[2; 0; 1]; C=[1,2,0]; D=0;
>> dbode(F, G, C, D, 0.1, 1)
```

这里的频率向量是依靠该函数自动选择的，绘制出来的频率响应曲线如图 5-4 所示。

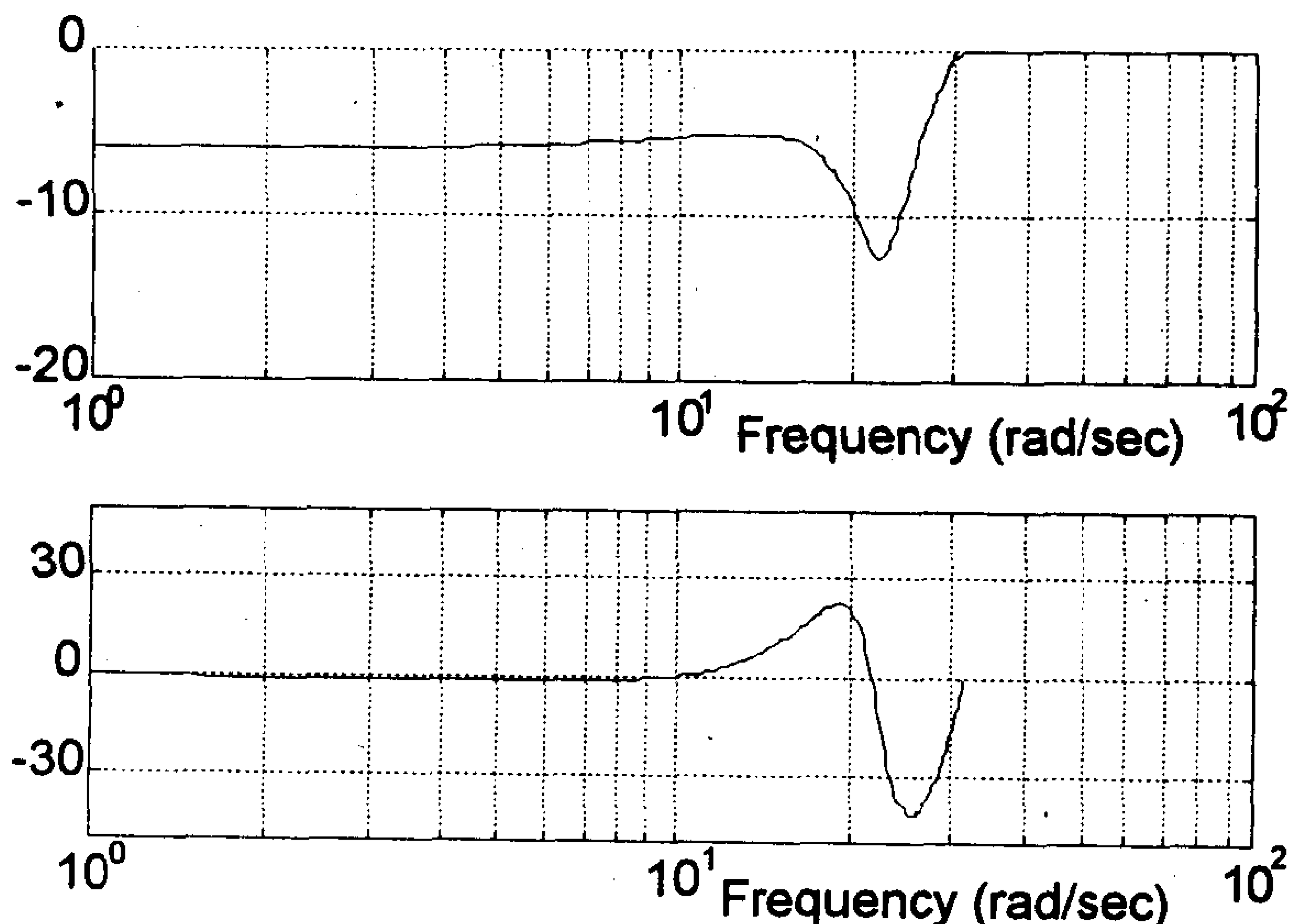


图 5-4　离散时间系统的 Bode 曲线

离散时间系统的频率响应曲线还可以由 dnyquist(), dnichols() 等函数直接绘制出来，它们的调用格式和其连续版本基本是一致的，在这里就不再叙述了。

5.1.4 时间延迟系统的频率响应

带有时间延迟的连续控制系统传递函数模型可以写成

$$G(s)=\frac{b_1s^m+b_2s^{m-1}+\cdots+b_{m+1}}{a_1s^n+a_2s^{n-1}+\cdots+a_ns+a_{n+1}}\mathrm{e}^{-Ts}=\widehat{G}(s)\mathrm{e}^{-Ts} \tag{5.1.7}$$

式中 T 为延迟时间常数，可以看出，带有时间延迟的系统从某种意义上说相当于在一个不带有时间延迟的传递函数模型 $\widehat{G}(s)$ 后面串接一个纯时间延迟环节 e^{-Ts}。带有时间延迟的状态方程模型可以写成

$$\dot{x}(t)=Ax(t)+Bu(t-T),\quad y=Cx(t)+Du(t-T) \tag{5.1.8}$$

其中 (A,B,C,D) 矩阵和不含有时间延迟的是类似的。对于给定的传递函数模型来说，该系统的频率响应可以由下面的方法直接求得，首先求出 $\widehat{G}(s)$ 的频率响应 (以幅值相位的形式表示)，这样整个系统的幅值频率响应和 $\widehat{G}(s)$ 的是一致的，而相位频率响应等于 $\widehat{G}(s)$ 的相位再减去 $-T\omega$，亦即

$$|G(j\omega)|=|\widehat{G}(j\omega)|,\quad \angle G(j\omega)=\angle\widehat{G}(j\omega)-T\omega \tag{5.1.9}$$

纯时间延迟环节 e^{-Ts} 可以由有理函数 (即传递函数类的分子分母多项式形式) 来近似，1892 年法国数学家 Padé 曾提出了一种著名的有理近似方法，后人将之命名为 Padé 近似方法，其表达式为

$$\mathrm{e}^{-Ts}\simeq\frac{1-Ts/2+p_1(Ts)^2-p_2(Ts)^3+p_3(Ts)^4-p_4(Ts)^5+p_5(Ts)^6-+\cdots}{1+Ts/2+p_1(Ts)^2+p_2(Ts)^3+p_3(Ts)^4+p_4(Ts)^5+p_5(Ts)^6+\cdots} \tag{5.1.10}$$

式中 6 阶以下的 Padé 近似系数 p_i 由表 5-1 给出，对于 $k>6$ 当然也可以求出相应的 Padé 近似系数，在这里就不将有关的系数列出了。控制系统工具箱提供了一个函数 pade() 来计算 Padé 近似的系数，它的调用格式为

```
[num, den]=pade(T, n)   或   [A, B, C, D]=pade(T, n)
```

其中 T 为延迟时间常数，n 为要求拟合的阶数，调用该函数之后将返回 Padé 近似的传递函数模型 num, den 或等效的状态方程模型 (A, B, C, D)。

由式 (5.1.10) 可以看出，Padé 近似的幅值在任何 T,s 下均恒等于 1，所以我们只对相位进行频率响应分析，并将结果在图 5-5 中显示出来，可以看出，低频段的拟合还是相当好的，高频段有时不是很理想。当然，拟合的阶次越高，得出的拟合精度也越高，但同时也会加大计算量。一般情况下，取 Padé 近似的拟合阶次为 3 或 4 就可以获得相当满意的精度。另外显而易见，在时间延迟常数较大时往往不如对小时间延迟的近似。

表5-1　Padé 近似系数表

阶次n	p_1	p_2	p_3	p_4	p_5
1	0	0	0	0	0
2	1/12	0	0	0	0
3	1/10	1/120	0	0	0
4	3/28	1/84	1/1680	0	0
5	1/9	1/72	1/1008	1/30240	0
6	5/44	1/66	1/792	1/15840	1/665280

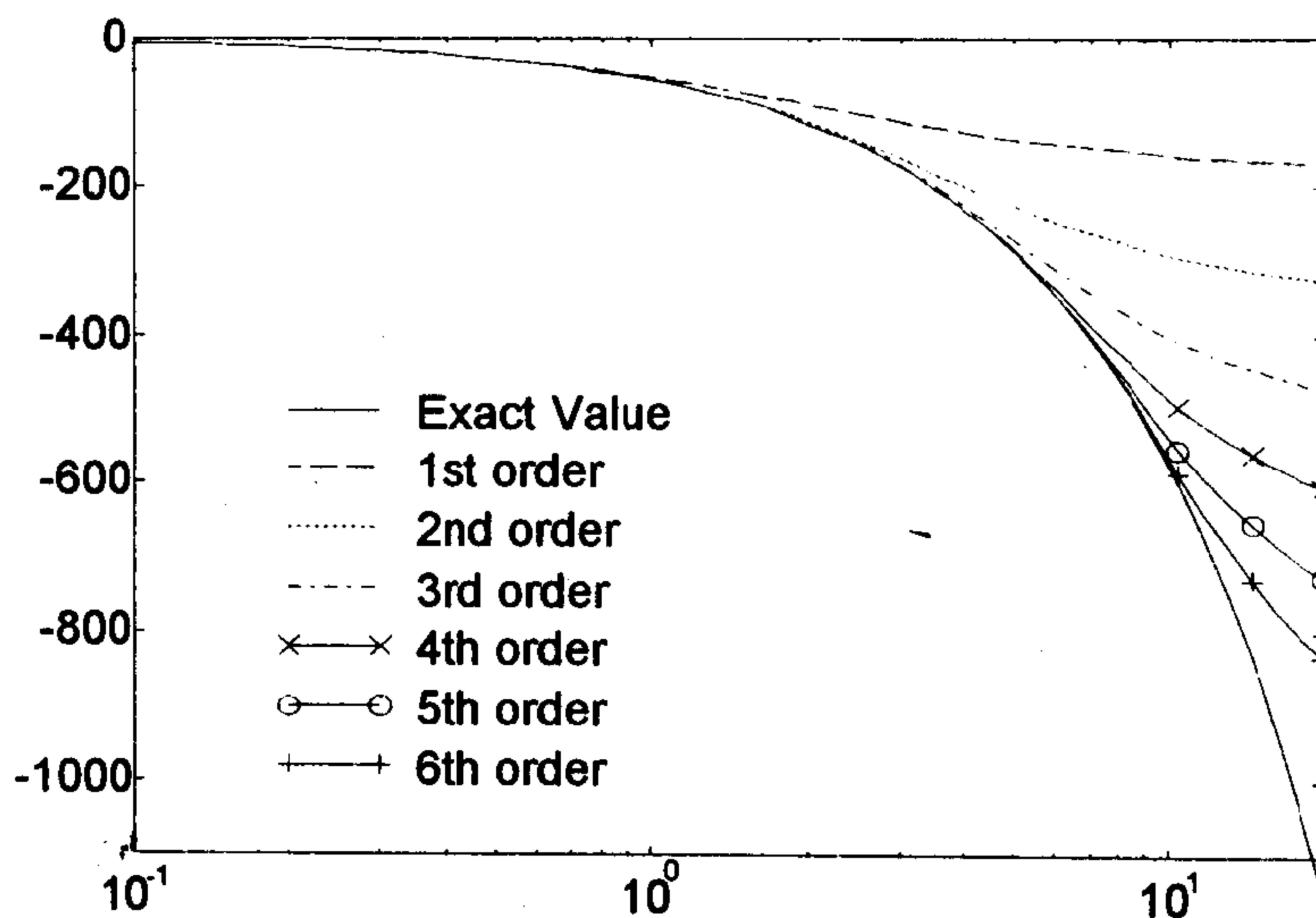


图5-5　时间延迟 Padé 近似的相位精度比较

例 5.3　考虑下面的传递函数模型

$$G(s)=\frac{s+1}{(s+2)^3}e^{-0.5s}$$

首先可以输入模型的有理传递函数并选择频率向量 ω，然后求出 $\widehat{G}(j\omega)$，再由式 (5.1.9) 和 4 阶 Padé 近似两种方法分别计算出系统的频率响应来

```
>> num=[1 1]; den=conv([1,2],conv([1,2],[1,2]));
>> w=logspace(-1,2); T=0.5;
>> [m1, p1]=bode(num, den, w);
>> p1=p1-T*w'*180/pi;
>> [n2,d2]=pade(T,4);
>> numT=conv(n2,num); denT=conv(den,d2);
>> [m2, p2]=bode(numT, denT, w);
>> subplot(211); semilogx(w,20*log10(m1),w,20*log10(m2),'--')
>> subplot(212); semilogx(w,p1,w,p2,'--')
```

由上面的程序段绘制的频率响应 Bode 曲线如图 5-6 所示，可见由这两种方法得出 Bode 图的幅

值曲线是一致的，在相位曲线上存在一些差异，尤其在高频部分更是如此。

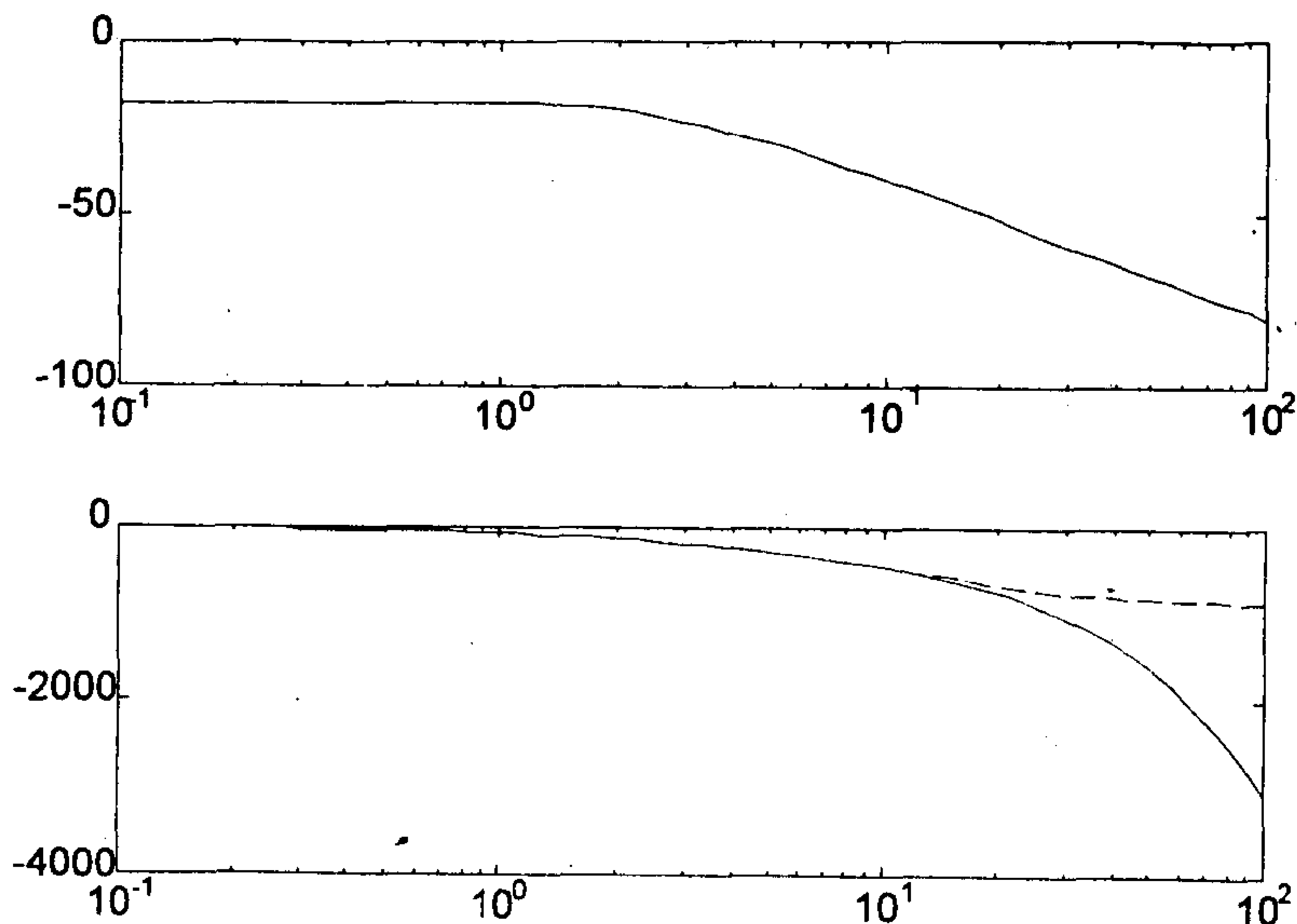


图5-6　时间延迟系统的 Bode 曲线

5.2　线性系统的时间响应分析

系统仿真实质上就是对描述系统的数学模型进行求解，对控制系统来说，系统的数学模型实际上是某种微分方程或差分方程模型，因而在仿真过程中需要以某种数值算法从给定的初始值出发，逐步地计算出每一个时刻系统的响应，最后绘制出系统的响应曲线，由此来分析系统的性能。在第 3 章中曾经介绍过一般常微分方程的数值解法，该方法是系统仿真的基础。其实对于第 4 章中介绍的各种线性系统模型来说，当然没有必要采用那些通用的算法来完成这种任务，而是应该充分地利用线性系统的特点，采取更简单的方法来得出问题的解，这样做不但会大大提高运算的效率，而且可以提高仿真的精度和可靠性。

MATLAB 的控制系统工具箱提供了很多线性系统在特定输入下仿真的函数，例如连续时间系统在阶跃输入激励下的仿真函数 step()，脉冲激励下的仿真函数 impulse() 及任意输入表激励下的仿真函数 lsim() 等，其中阶跃响应函数 step() 的调用格式为

```
[y, x]=step(num,den,t)
```

或

```
[y, x]=step(A, B, C, D, iu, t)
```

其中第一种调用方式下 num 和 den 分别为线性系统传递函数模型的分子和分母多项式系数，t 为选定的仿真时间向量。此函数的返回值 y 为系统在各个仿真时刻的输出所组成的矩阵，而 x 为自动选择的状态变量的时间响应数据 [1]，同样一个函数还可以由不同的方式来调用，这里如果给出系统的状态方程模型 (A, B, C, D)，并指定输入变量的序号 iu，则同样可以计算出系统的阶跃响应数据。类似于前面介绍的 bode() 函数，如果用

[1] 这里系统的状态变量是依据第 4 章中的可控标准型实现而自动选定的。

户对具体的响应数值不感兴趣，而只想绘制出系统的阶跃响应曲线，则可以由如下的格式调用此函数来完成：

```
step(num,den,t);    或   step(A, B, C, D, iu, t);
step(num,den);      或   step(A, B, C, D, iu);
```

例 5.4　假设系统的开环传递函数模型为

$$G(s)=\frac{20}{s^4+8s^3+36s^2+40s}$$

试求出该系统在单位负反馈下的阶跃响应曲线。要求解这样的问题，首先应该得出系统的闭环传递函数模型，然后再对之进行仿真分析。这一任务的 MATLAB 实现为

```
>> numo=20;  deno=[1,8,36,40,0]; t=0:.1:10;
>> numc=numo; denc=[zeros(1,length(deno)-length(numo)),numo]+deno;
>> [y,x]=step(numc,denc,t); plot(t,y,t,x)
```

这样得出的输出及状态变量的时间响应图形如图 5-7 所示。这一问题还可以首先转换成状态方程模型，然后调用 step() 函数来求解系统的响应

```
>> [A, B, C, D]=tf2ss(numc, denc);
>> [y, x]=step(A, B, C, D, 1, t);
>> plot(t,y,t,x)
```

当然这样的仿真结果和前面得出的是完全一致的。

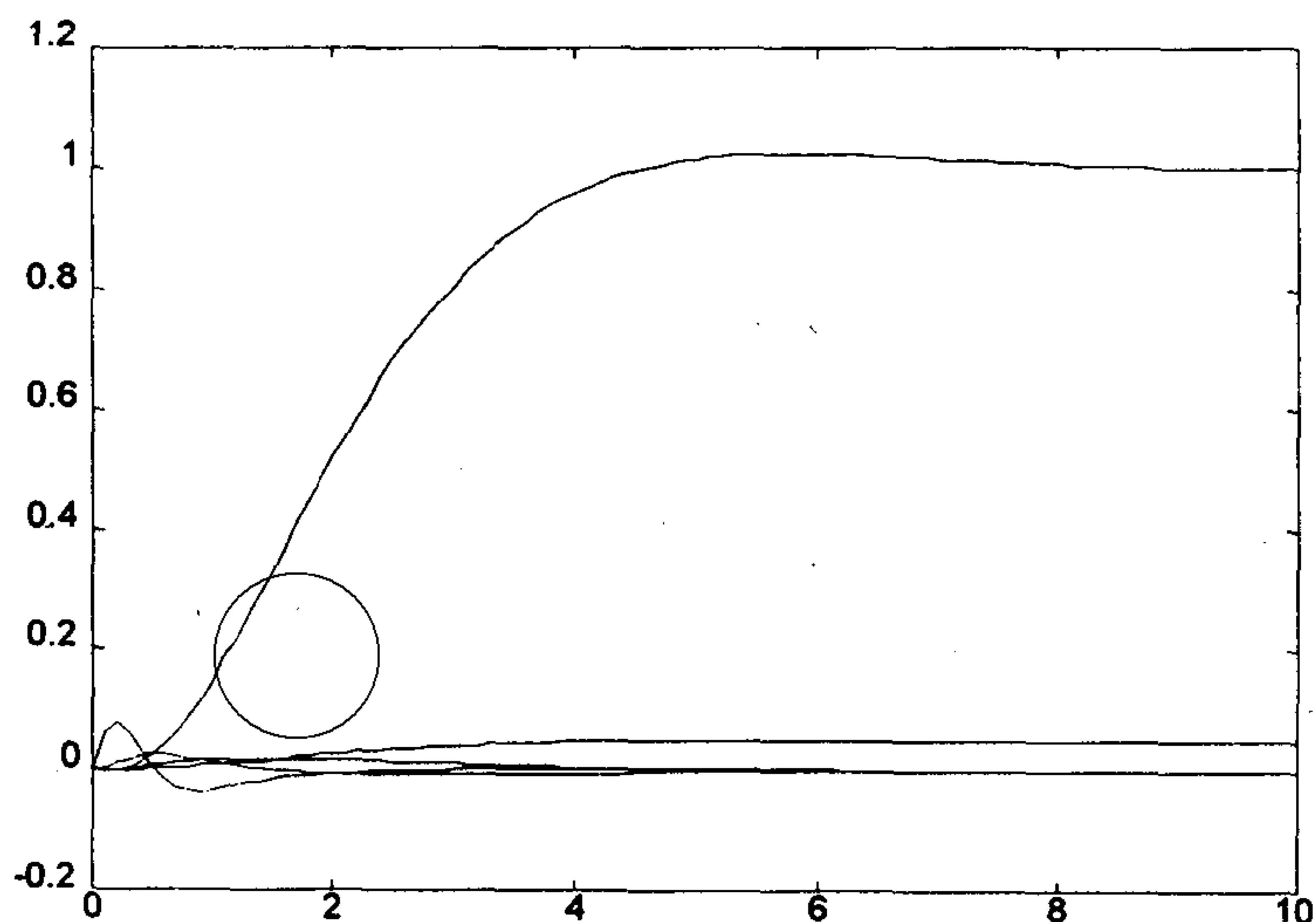


图 5-7　控制系统的仿真结果

求取脉冲响应的函数 impulse() 和 step() 函数的调用格式是完全一致的，而任意输入下的仿真函数 lsim() 的调用格式稍有不同，因为在此函数的调用时还应该给出一个输入表向量，该函数的调用格式为



```
[y, x]=lsim(num, den, u, t)
```
或
```
[y, x]=lsim(A, B, C, D, iu, u, t)
```

式中 u 为给定输入构成的列向量，它的元素个数应该和 t 的个数是一致的。当然该函数若调用时不返回参数，则也可以直接绘制出响应曲线图形。

MATLAB 还提供了离散时间系统的仿真函数，包括阶跃响应函数 dstep()，脉冲响应函数 dimpulse() 和任意输入响应函数 dlsim() 等，它们的调用方式和连续系统的不完全一致，例如阶跃响应函数的调用格式为

```
[y, x]=dstep(num, den, n);
```

这里 num，den 分别为脉冲传递函数的分子和分母系数向量，而 n 为需要仿真的采样点个数。同样地用户还可以很容易地使用和掌握 dimpulse() 和 dlsim() 函数的功能和调用方法。

例 5.5　延迟系统的时间响应也可以用 Padé 方法来近似地求出，例如考虑例 5.3 中给出的系统模型，由下列 MATLAB 语句可以分别绘制出在不同 Padé 近似阶次下原系统与近似系统阶跃响应比较的曲线，如图 5-8 所示。

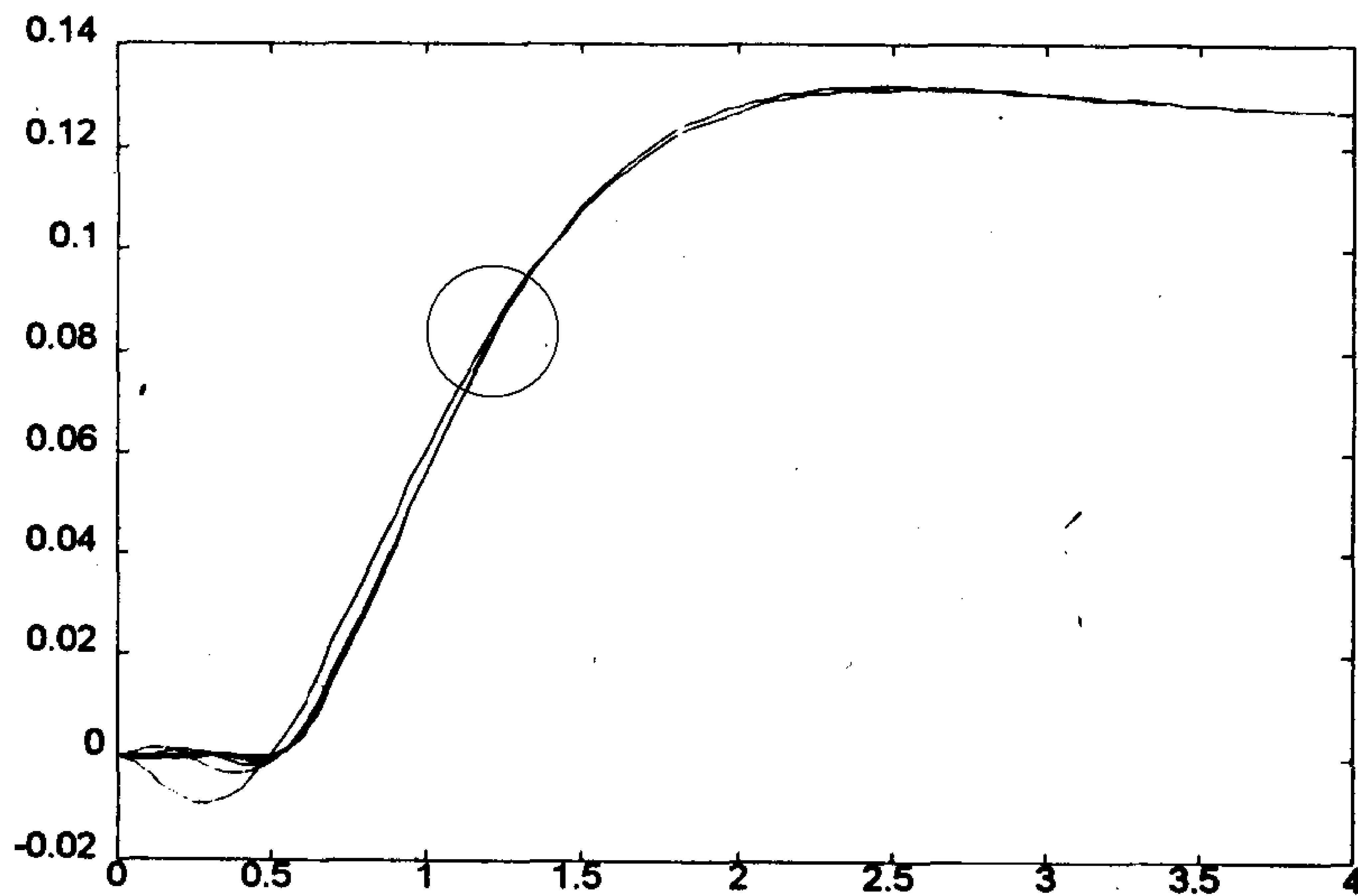


图5-8　不同阶次下的近似阶跃响应

```
>> t=0:.05:4;  y=step(num,den,t); T=0.5;
>> ii=find(t>=T);
>> y=[zeros(ii(1)-1,1); y(1:length(t)-ii(1)+1)];
>> for i=1:6
      [np,dp]=pade(T,i);
      nn=conv(num,np); dd=conv(den,dp);
      y=[y step(nn,dd,t)];
   end
>> plot(t,y)
```

在上面段落的前一个部分求出不带有时间延迟系统的阶跃响应，并根据延迟的时间常数在该数据的前面补上若干个 0，这样就得出时间延迟系统阶跃响应的精确解，得出该精确解之后，还可以由各阶 Padé 近似的方法得出延迟系统阶跃响应的近似解，并全部存入 y 变量，最后调用 plot() 函数将得出的响应绘制出来。从图 5-8 可以看出，各阶 Padé 近似所得出的结果都是比较接近原系统的，随着拟合阶次的增高，拟合的精度也会有所改进。

若系统的开环模型为带有纯时间延迟的结构 $G(s)=B(s)\mathrm{e}^{-Ts}/A(s)$，则单位负反馈系统可以由下面的算法进行近似，首先写出系统的闭环模型

$$G_{闭环}(s)=\frac{N(s)\mathrm{e}^{-Ts}}{D(s)+N(s)\mathrm{e}^{-Ts}} \tag{5.2.1}$$

然后将闭环系统分母中的时间延迟 e^{-Ts} 用 Padé 表达式来近似，即 $\mathrm{e}^{-Ts}\simeq N_p(s)/D_p(s)$，并保持系统分子中的延迟环节不变，这时闭环系统的模型可以近似成

$$G_{闭环}(s)\simeq\frac{N(s)D_p(s)}{D(s)D_p(s)+N(s)N_p(s)}\mathrm{e}^{-Ts} \tag{5.2.2}$$

这样得出的系统又可以由简单的延迟系统的形式来近似描述了，下面将通过例子来演示这样近似得出的近似精度。

例 5.6　将例 5.3 中给出的模型当作开环模型，若选 Padé 近似的阶次为 3，则利用上面的近似规则写出如下的 MATLAB 语句

```
>> [np,dp]=pade(T,3);
>> numc=conv(num,dp)
numc = 0.0010    0.0253    0.2667    1.2123    0.9699
>> den0=conv(den,dp); num0=conv(np,num);
>> denc=[zeros(1,length(den0)-length(num0)) num0]+den0
denc = 0.0010    0.0303    0.3980    2.7702    8.4862   15.0328    9.6986
>> y=step(numc,denc,t);
>> ii=find(t>=T);
>> y=[zeros(ii(1)-1,1); y(1:length(t)-ii(1)+1)];
>> plot(t,y)
```

最后得出的结果如图 5-9 所示。在图中除了得出的近似解外还绘制出了系统的精确解，而这一精确解是由 SIMULINK 软件得出的，在下面各节中将介绍 SIMULINK 工具的使用方法。从图 5-9 可以看出，采用这种近似是可以由较高的精度对系统进行仿真的。可以验证，对此系统来说采用这样的近似方法比纯 Padé 近似的方法精度要高。

5.3 系统框图输入与仿真工具 SIMULINK

如果控制系统的结构很复杂，例如如果想仿真第 4 章中介绍过的 F-14 飞机模型，则若不借助专用的系统建模软件 (如 BLOCKM [11] 或 BlockEdit [5])，在过去很难准确地把一个控制系统的复杂模型输入给计算机，然后对之进行进一步地分析与仿真。1990

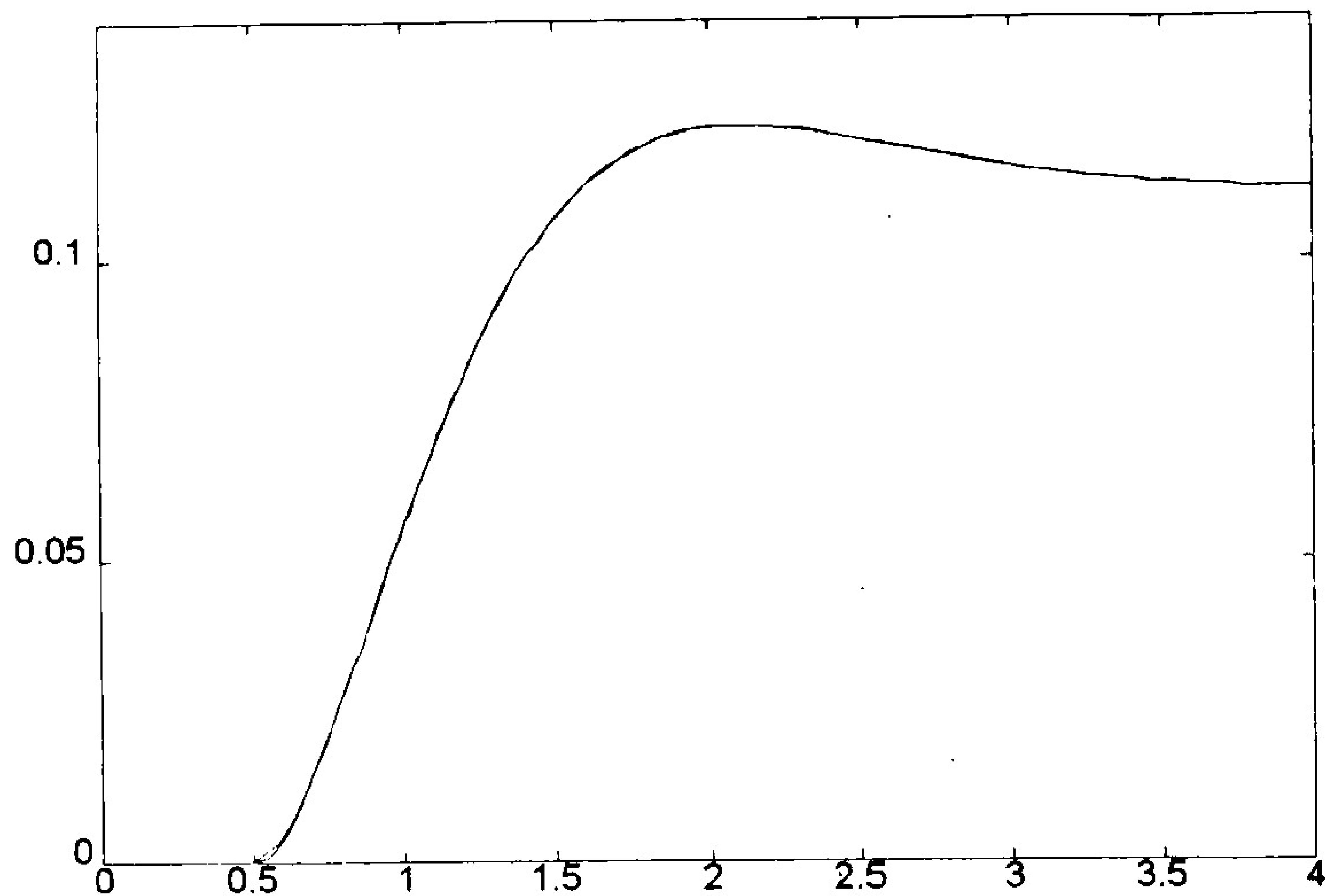


图5-9 延迟闭环控制系统的仿真结果

年 MathWorks 软件公司为 MATLAB 提供了新的控制系统模型图形输入与仿真工具，并定名为 SIMULAB，该工具很快在控制界就有了广泛的使用。但因其名字与著名的 SIMULA 软件类似，所以 1992 年以来正式改名为 SIMULINK，这一名字的含义是相当直观的，因为它较明显地表明此软件的两个显著的功能： SIMU(仿真) 与 LINK(连接)，亦即可以利用鼠标器在模型窗口上“画”出所需的控制系统模型，然后利用 SIMULINK 提供的功能来对系统进行仿真或线性化分析。这种做法的一个优点是，可以使得一个很复杂系统的输入变得相当容易且直观。

5.3.1 控制系统框图模型建立

进入 MATLAB 环境之后，键入 simulink 命令则可以打开相应的系统模型库如图 5-10 所示，这一模型库包括以下各个子模型库： Sources (输入源)、Sinks (输出方式)、

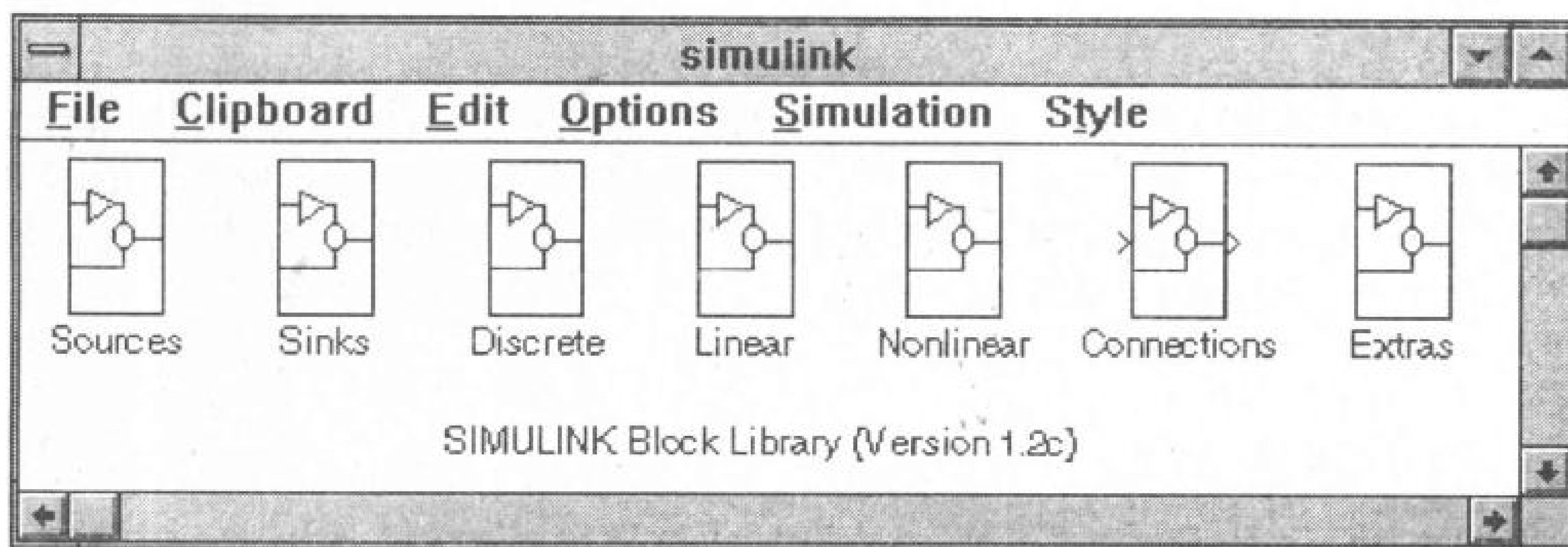


图5-10 SIMULINK 的程序界面

Discrete(离散时间模型)、Linear (线性环节)、Nonlinear (非线性环节)、Connections

(连接及接口)、Extras (其它环节)。

若想建立一个控制系统结构框图，则应该选择 File | New 菜单项，这样就会自动打开一个空白的模型编辑窗口，允许用户输入自己的模型框图。打开输入源模块库的图标，将出现一个如图 5-11(a) 所示的子模块库，可见此模块库中包含下列各个子模块：阶跃

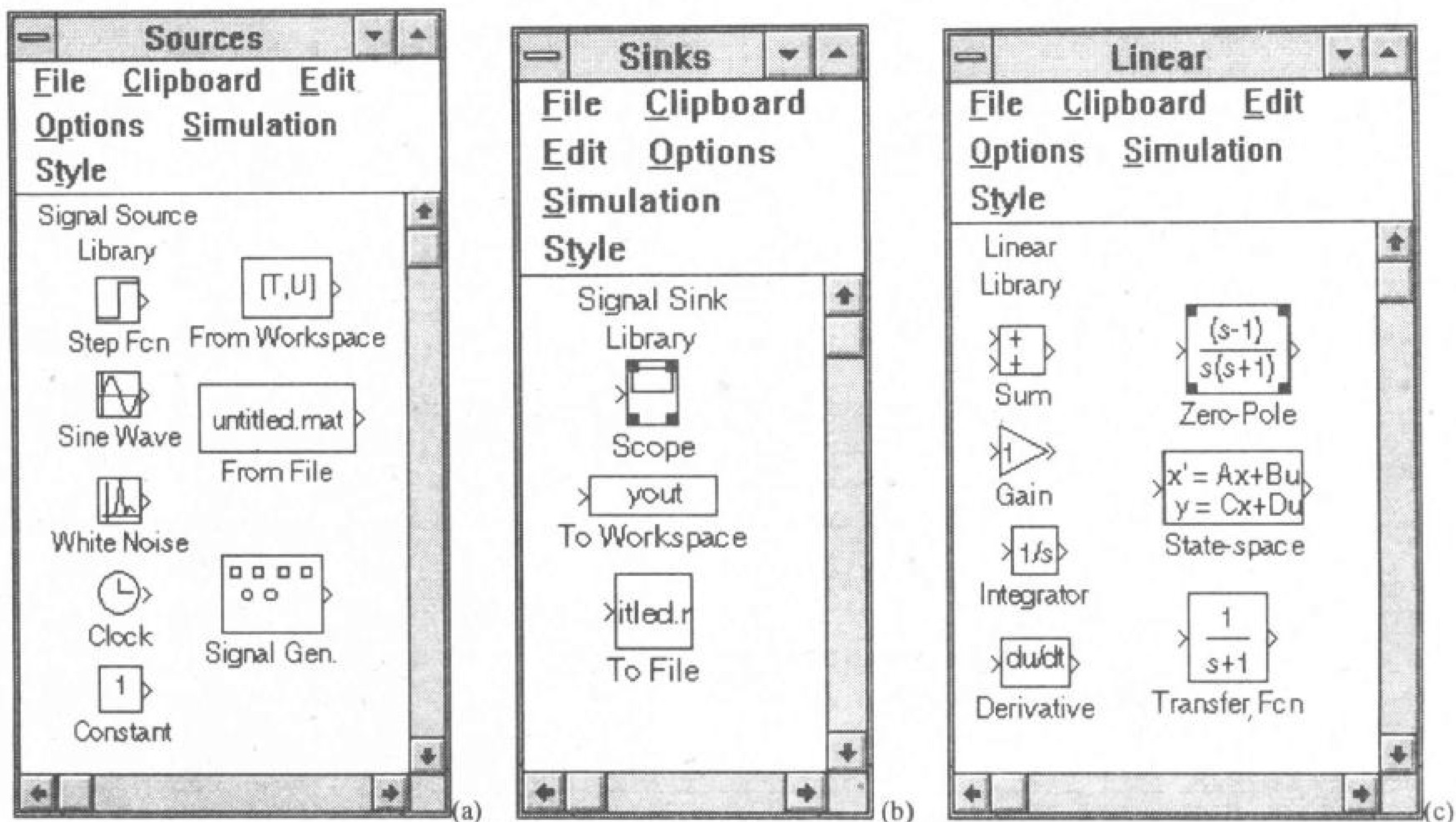


图5-11 SIMULINK 中的常用子模型库

函数、正弦函数、白噪声函数、时钟[1]、常数、MATLAB 空间变量、信号发生器等，可以利用鼠标点中的方式来选择所需的子模块[2]，并将之拖动到所打开的模型窗口上。打开其中的 Linear (线性) 和 Sinks (输出) 各个图标也会给出相应的子模块库，如图 5-11(b), (c) 所示，而这些模块都可以采用选中后拖动的方法复制到模型窗口中去。连接两个模块是相当容易的，因为可以简单地用鼠标先点一下起点模块的输出端 (三角符号)，然后拖动鼠标器，这时就会出现一条带箭头的直线，将它的箭头拉到终点模块的输入端再释放鼠标键，则 SIMULINK 会自动地产生一条带箭头的连线，将两个模块连接起来。

例 5.7　例如若想建立起一个如图 5-12 所示的典型 PID 控制系统模型，则首先应该调用 SIMULINK 界面的 File | New 菜单项来打开一个空白的编辑窗口，准备绘制系统的方框图。打开一个新的编辑窗口之后，可以在该窗口内建立起阶跃输入模型，而这又应该选择图 5-11(a) 中给出的输入模块库中的阶跃输入模块 (其标注为 step)，将之用鼠标器拖动到编辑窗口中后释放鼠标键，这时用户就会

[1]因为一般采用变步长的仿真方法，所以这一输入源也是相当重要的。
[2]一个模块选中之后，在它的周围将出现黑点来标注。

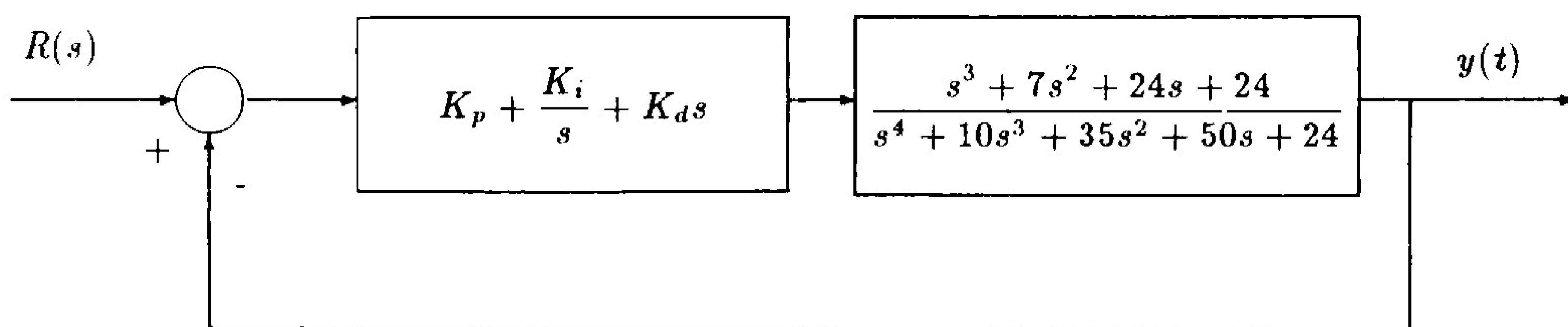


图5-12 典型 PID 控制系统的方框图

发现在该窗口内出现了一个阶跃输入的图标。此外因为可能采用变步长的仿真方法，所以还应该将图 5-11(a) 中给出的时钟输入图标也拖动到编辑窗口内然后释放鼠标键，这样在编辑窗口内就会出现一个时钟的图标。

各个模块的参数是可以随意改动的，比如说 SIMULINK 给定的阶跃输入模块的跳跃时间是在 1 时刻，而我们经常使用的是在 0 时刻，这样就需要对其设置进行改动。首先应该用鼠标器左键双点阶跃输入模块图标，这样将得出一个如图 5-13 所示的对话框， 用户可以在该对话框中修改其中的

Step Fcn

Block name: Step Fcn

Block type: Step Fcn

At t = Step time, output changes from Initial value to Final Value.

OK

Cancel

Help

Step time:

1

Initial value:

0

Final value:

1

图5-13 阶跃输入信号参数修改对话框

Step time (阶跃时刻) 后面的编辑框来改变有关的参数，当然用户还可以修改 Initial Value (初始值) 和 Final Value (终止值) 引导的编辑框来重新定义阶跃信号。除了阶跃和时钟输入之外， SIMULINK 还允许用户输入正弦波、白噪声等其它输入方式，此外还可以使用 SIMULINK 提供的信号发生器 (Signal Gen) 来给出输入信号，双点该图标将给出如图 5-14 所示的对话框，可见该模块中提供了 4 种输入方式：正弦、方波、锯齿波和随机输入，用户可以从中选择一种输入信号，并指定有关的特殊参数即可。在每种输入方式下都有自己对应的特殊参数，用户当然可以采用双点该图标的方式来修改有关的参数，因为在每个参数输入之前 SIMULINK 相应的模块都给出了较实用的提示，所以用户可以通过自己摸索的方式来学习每一个模块的使用方法。

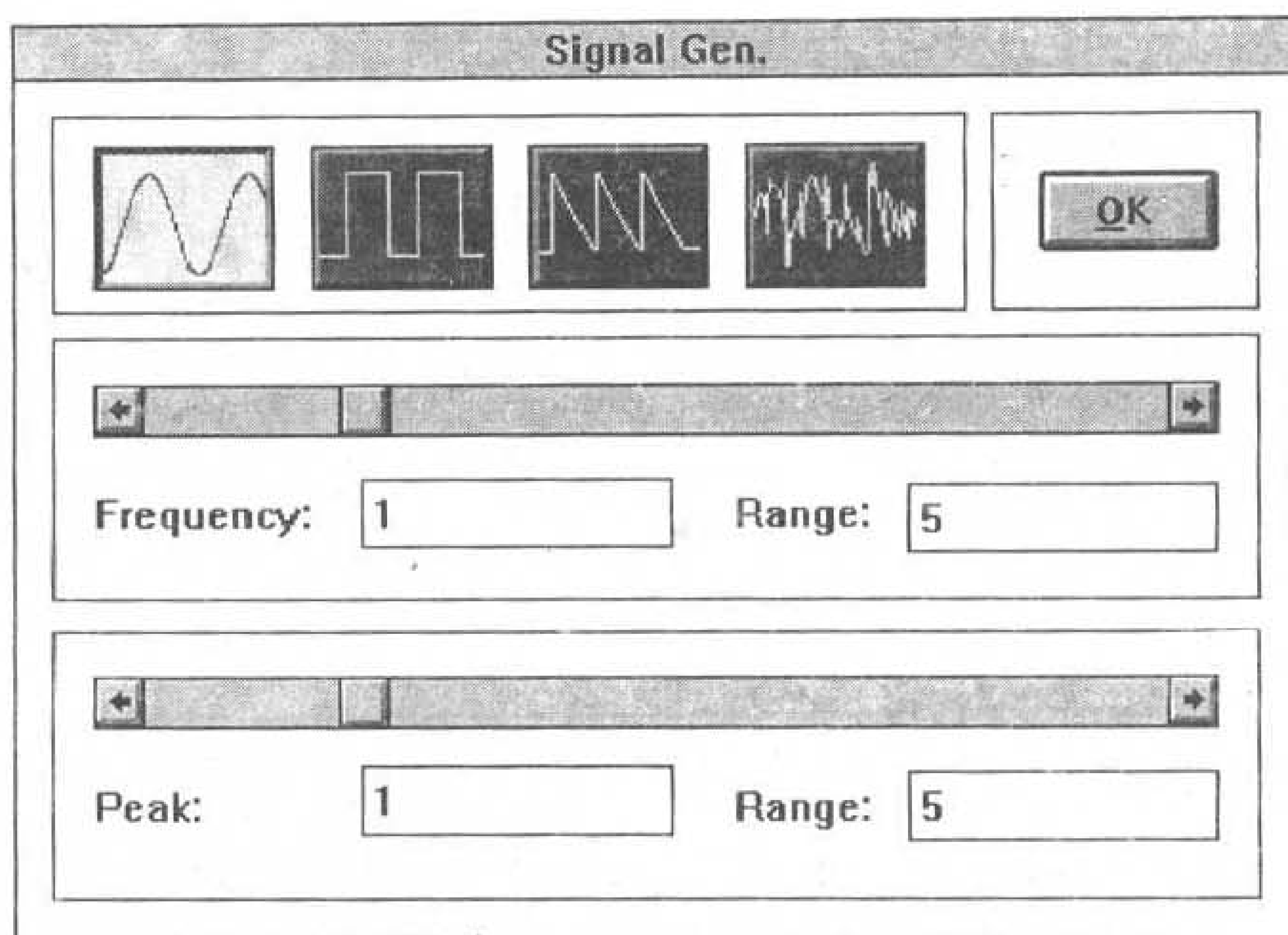


图5-14　信号发生器对话框

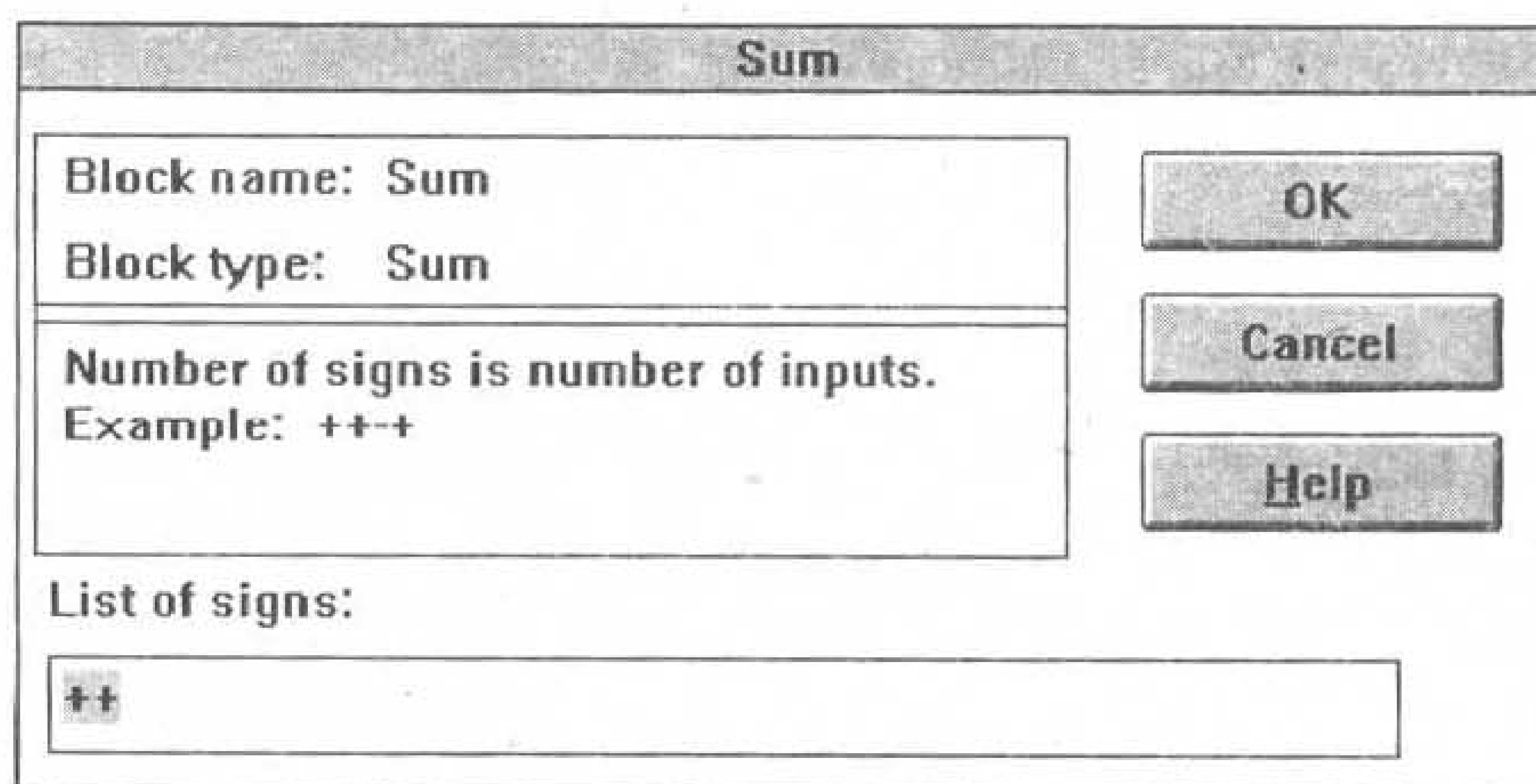


图5-15　加法器参数修正对话框

因为在系统中有负反馈环节，所以应该在模型窗口中加入一个加法器，这就需要首先打开 Linear 模块库，然后从中选中加法器 (Sum) 模块的图标，并将之拖动到模型窗口中去，再释放鼠标键。因为这里使用的是单位负反馈，所以需要对加法器中的符号 (默认值为 ++) 进行修正，具体方法是双点该图标来获得如图 5-15 所示的对话框，在 List of Signs (符号列表) 引导的编辑框中键入 +- 字样，这时新的加法器的两个输入信号将为一正一负。下面还要将 PID 控制器的各个元件输入到模块窗口中去，这些元件都可以从线性模块库中选择出来。PID 控制器的各个环节中都有一个比例模块，用户可以通过双点相应图标的方法来获得参数修正对话框，如图 5-16 所示，容易地修改比例系数 (Gain)，注意，在输入数据时除了直接填写数值外还可以填写 MATLAB 变量名。

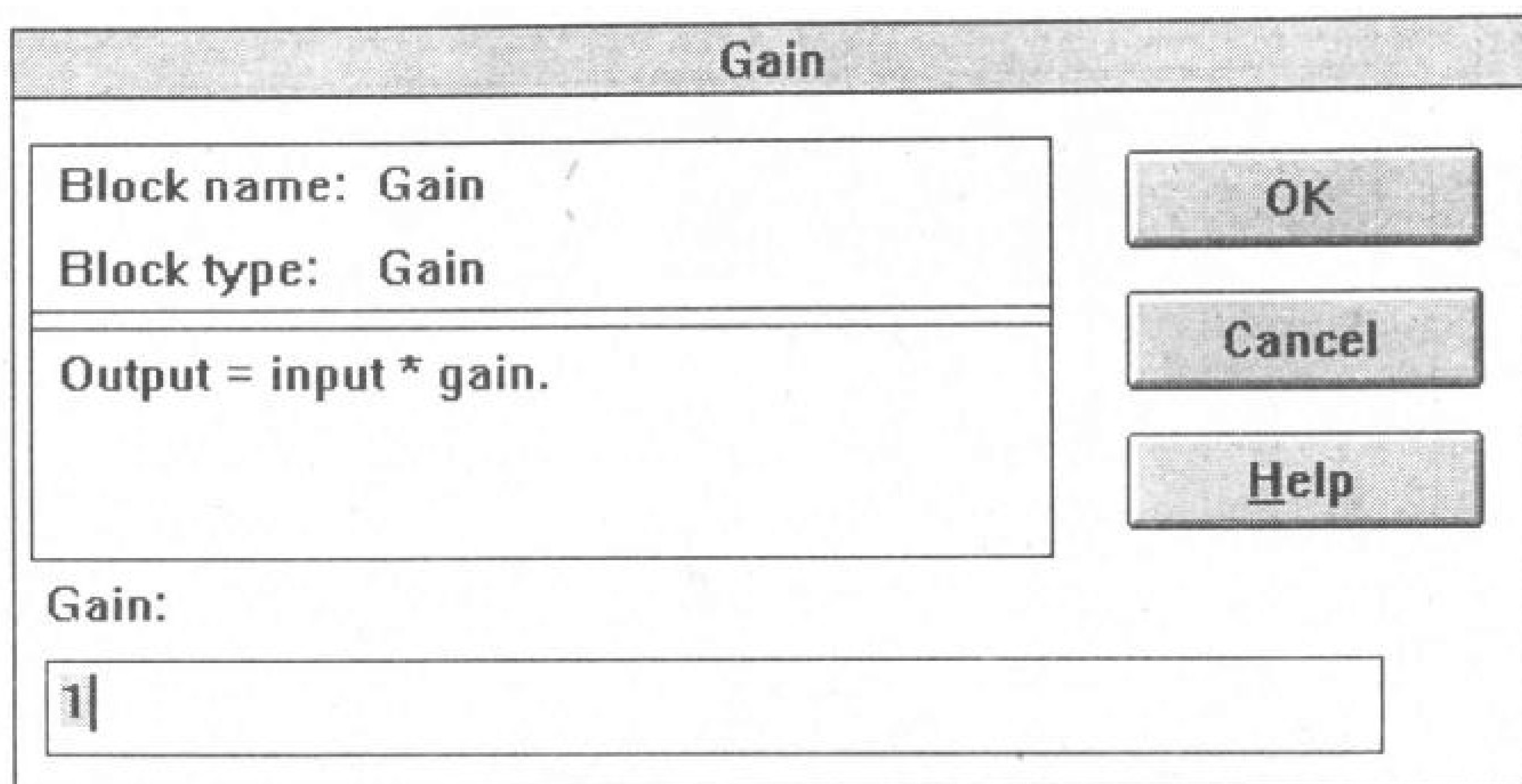


图5-16　比例模块参数修正对话框

双点积分模块的图标可以得出一个如图 5-17 所示的对话框，用户可以从中修改积分环节的初值 (Initial Value)。在 PID 控制器的后面还应该添加一个加法器，使得 3 个输出信号能够叠加起来。同样还需要对其中的符号列表进行修正，这时只需在图 5-15 所示的对话框中填写上 +++ 即可。

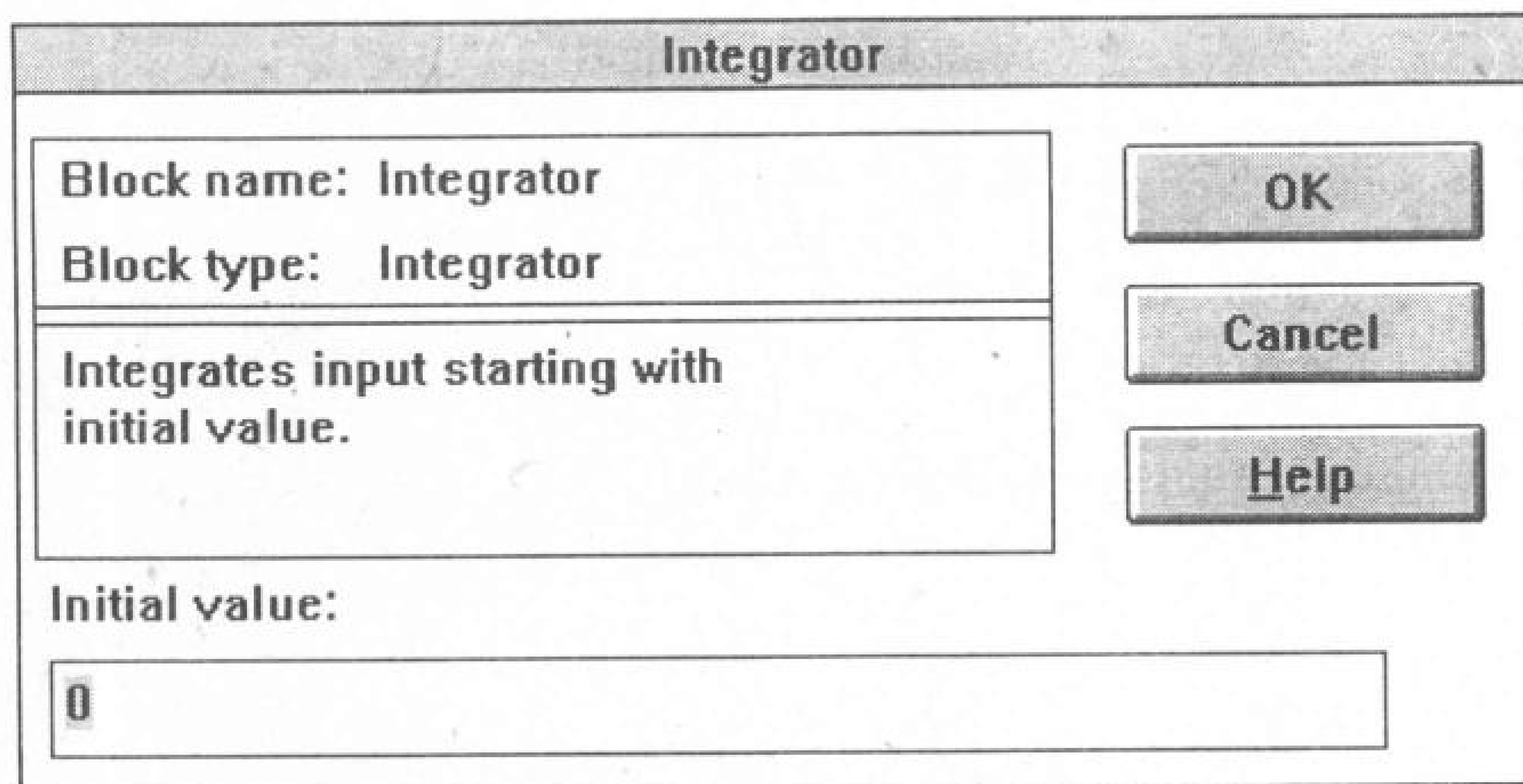


图5-17　积分模块参数修改对话框

对象模型是由传递函数给出的，所以应该在 Linear 模块库中选择传递函数模块的图标，并将之拖动到模型窗口中。这里的传递函数的初始值为 $1/(s+1)$。如果想改变其参数，则应该双点该图标来获得一个如图 5-18 所示的对话框，并分别在 Numerator (分子) 和 Denominator (分母) 引导的编辑框中填写系统传递函数的分子和分母多项式系数，然后选择 OK 按钮，这时该模块的值，以及该模块的图标显示都将赋予新的传递函数表示。

有了这些基本模块之后，还要引入输出模块，输出的各个模块是在 Sinks 模块库中给出的，打开该模块库将得到如图 5-11(b) 所示的各个子模块，从中选择 Scope (示波器) 和 To Workspace (传送到 MATLAB 工作空间) 两个模块，分别拖动到模型窗口中去，打开 Scope 模块，则可以得出如图 5-19 所示的示波器显示，用户可以根据自己的需要设定示波器的横纵坐标的范围，使得输出的结果能够在示波器上较好地显示出来。从图中可以看出， SIMULINK 提供的示波器和硬件示波器的效果

是很接近的。除了示波器形象的输出之外，用户还可以使用 To Workspace 模块将仿真结果返回到 MATLAB 的工作空间，这样返回的结果当然可以利用前面叙述的 MATLAB 命令来进行进一步处理，比如用 plot() 函数将结果绘制出来。向 MATLAB 工作空间传送数据时，应该给数据指定一个

Transfer Fcn

Block name: Transfer Fcn

Block type: Transfer Fcn

Vector expressions for numerator and denominator. Coefficients are in descending powers of s.

OK

Cancel

Help

Numerator:

[1]

Denominator:

[1 1]

图 5-18 传递函数参数修改对话框

Scope

Horizontal Range: 1

OK

Vertical Range: 1

图 5-19 示波器模块的显示界面

变量名，它是通过双点 To Workspace 模块的图标来完成的，这将得出一个如图 5-20 所示的对话框，用户可以在 Variable name (变量名) 引导的编辑框中输入相应的变量名，此外如果计算出来的数据

太多，也可以改变 Maximum number of rows (输出点的最大个数) 引导的编辑框来进行设置，在一般情况下，该参数选择为 1000 也就足够了。

To Workspace

Block name: To Workspace
Block type: To Workspace

Puts specified matrix in workspace.
Matrix has one column per input,
one row per simulation step.

OK

Cancel

Help

Variable name:

yout

Maximum number of rows (timesteps):

1000

图5-20　To Workspace 模块的参数设置对话框

按照上述的方法将所有各个模块画出来之后，就可以采用前面介绍的方法将相关的模块用鼠标器连接起来构成一个原系统的框图描述，亦即可以得出一个如图 5-21 所示的 PID 控制系统的 SIMULINK 描述，所以这样得出的模型就可以由 MATLAB 环境来直接处理了。

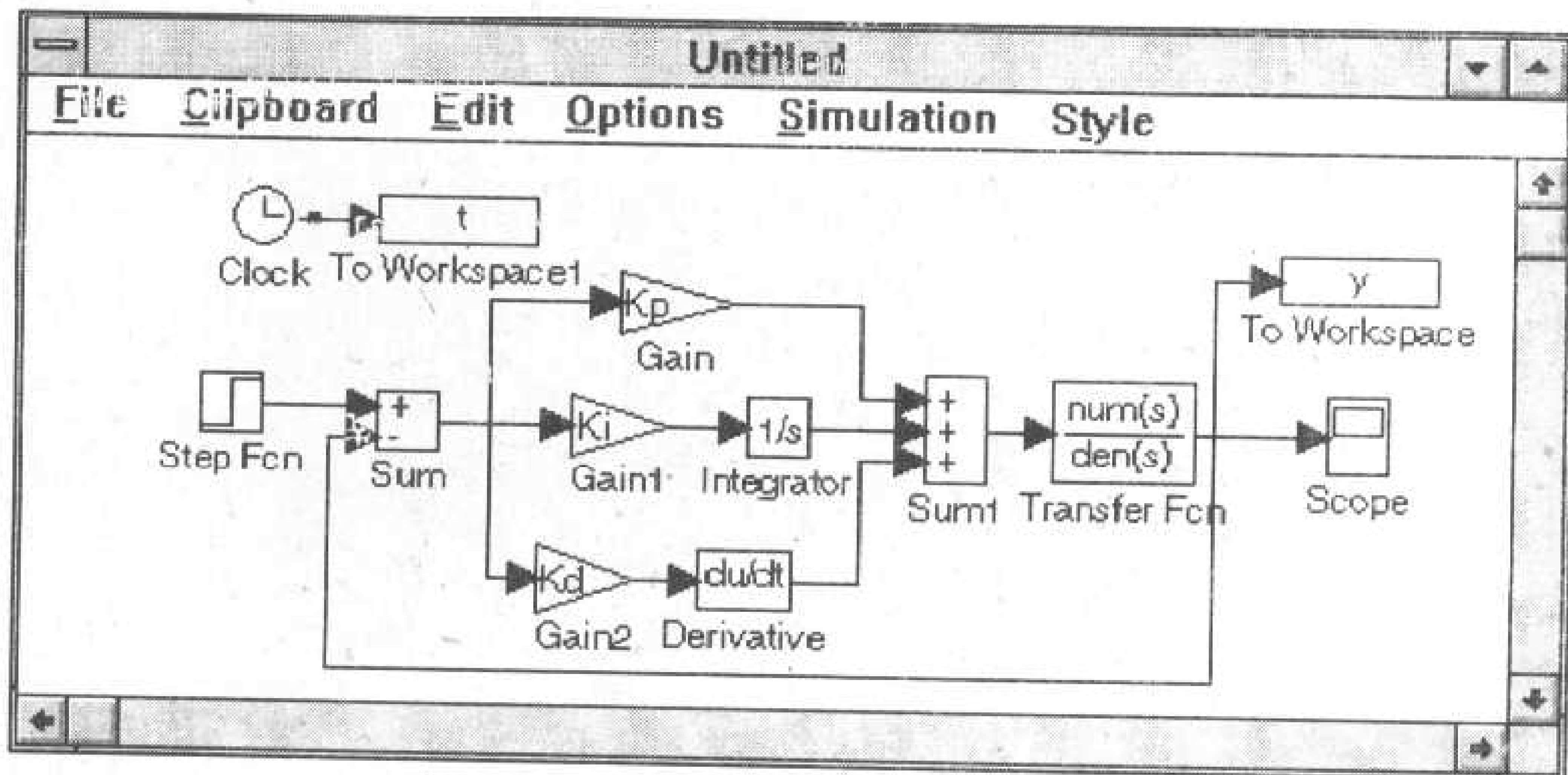


图 5-21　PID 控制系统的 SIMULINK 实现

5.3.2 利用 SIMULINK 进行数字仿真

建立起来系统模型之后，可以打开 Simulation (仿真分析) 菜单，这时将得出如图 5-22 所示的菜单结构， 在启动仿真过程之前，首先应选择 Simulation | Parameters 选项

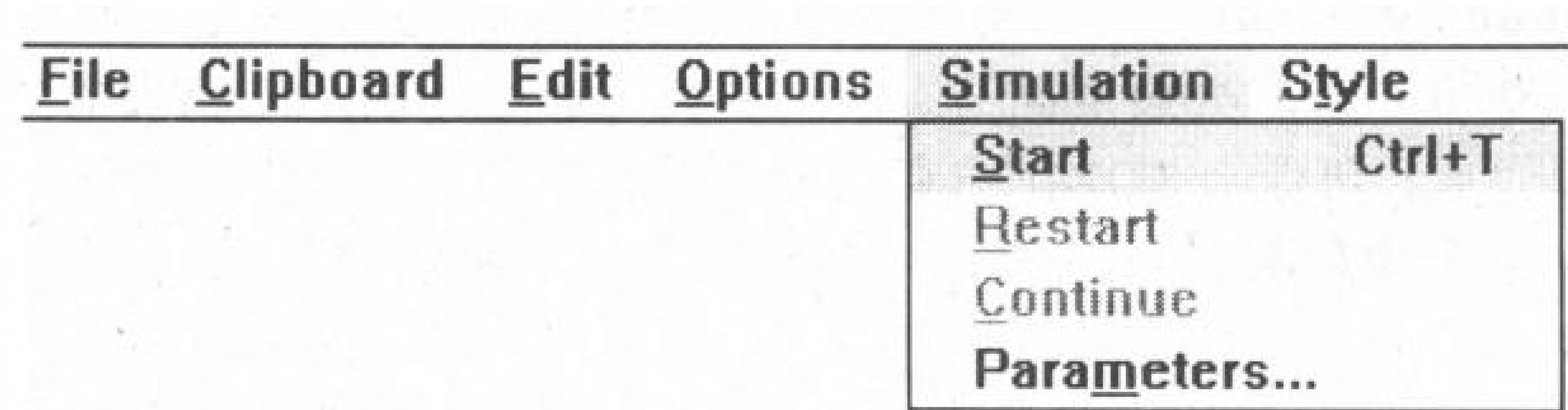


图5-22 SIMULINK 的 Simulation 菜单

来设置仿真控制参数，这时将给出一个如图 5-23 所示的对话框，如必要的话用户可以随意改变该对话框的默认内容，在对话框中定义了下述的有关仿真控制参数：

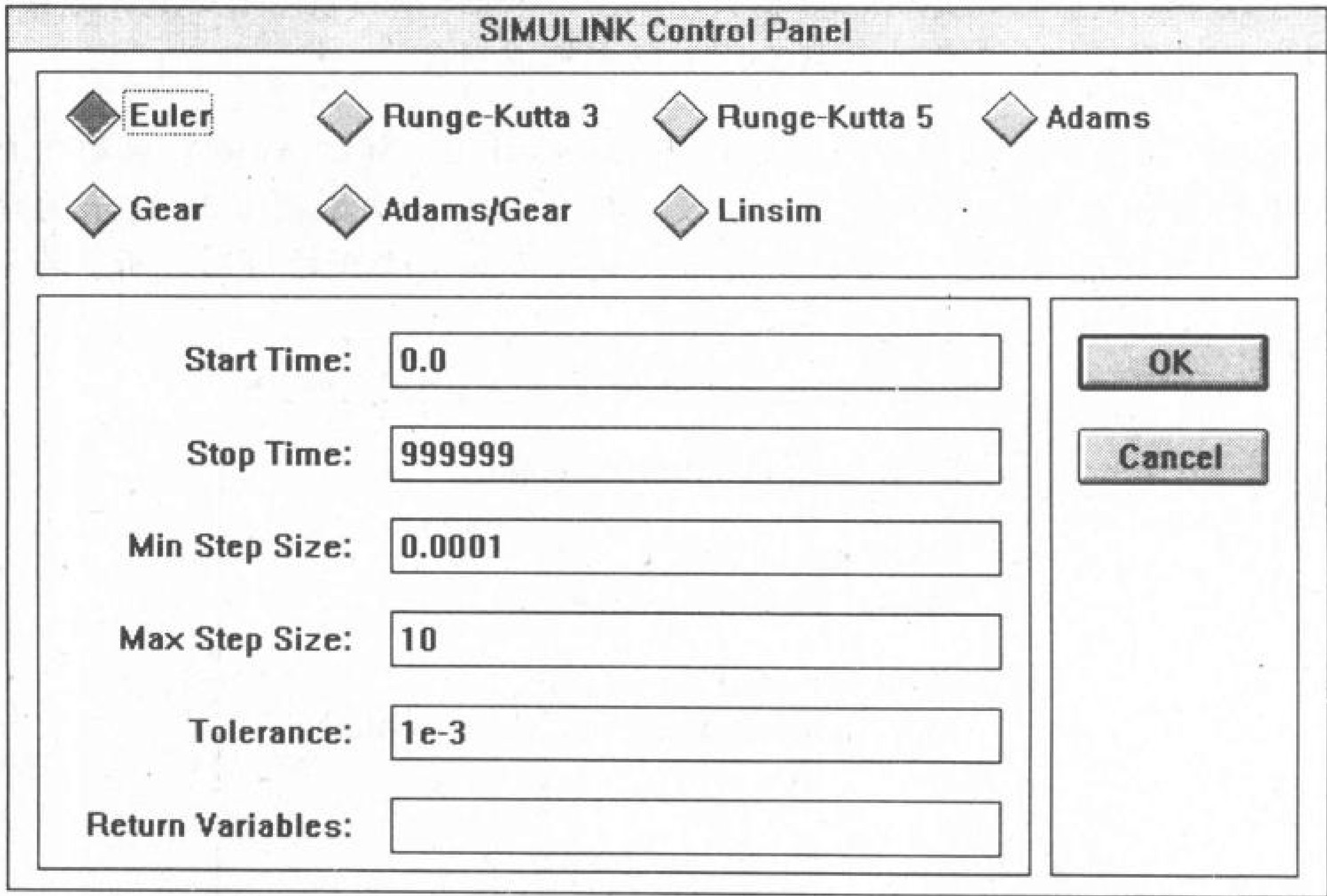


图5-23 仿真控制参数设置对话框

- **仿真算法的选择**: SIMULINK 提供了多种仿真用的数值算法，包括最简单的 (也是精度最低的)Euler 算法、2/3 阶和 4/5 阶 Runge-Kutta 算法、Gear 预报校正算法 (可以用来解决刚性问题) 、 Adams 预报校正算法以及专门适合于线性系统仿真的 LinSim 算法等，这些算法的适用范围是不同的，所以用户在解决实际问题时应该采用针对该问题的合适算法。

- **仿真范围的指定**: 由图 5-23 给出的对话框可以容易地看出，仿真范围是根据 Start Time (开始时间) 和 Stop Time (终止时间) 引导的编辑框来指出的，所以用户只需在这两个编辑框中填写上合适的数据就可以了。
- **仿真步长范围的指定**: 允许用户选择最大的仿真步长 (maximum step size) 和最小仿真步长 (minimum step size), 并采用了变步长的算法来比较合理地在给定的最大 / 最小范围内选择仿真步长。一般情况下，最大步长可以选择成一个较大的数值，如果最大步长选择得过大，可能会出现在仿真点处仿真结果是正确的，但仿真曲线不是很光滑的情况，故在选择最大步长时把它选择成整个仿真范围的 1/50 。最小步长如果选择得太小会增大计算量，尤其对 Adams 或 Gear 算法来说，由于最小步长并不能影响计算的精度，所以应该把最大或最小步长设定在能够得出光滑的曲线或利于分析的大小就可以了，而没有必要将之设置得过小来加大运算量。

  在一些特定的场合下往往还需要采用定步长的方法进行仿真，这时需要将仿真的最小步长设置成等于最大步长即可。
- **仿真精度的定义**: 在采用变步长算法时，应该先指定一个容许误差限，使得当误差超过这一误差限时会自动地对仿真步长作适当的修正，所以说，在变步长仿真时误差限的设置是很重要的，它将关系到微分方程求解的精度。

设置完仿真控制参数后，则可以选择 Simulation | Start 选项来启动仿真过程，这时如果仿真模型中有些参数没有定义，则会给出一个如图 5-24 所示的消息框来通知用户， 如果模型中所有参数均有定义，则可以正式开始仿真分析。若依靠示波

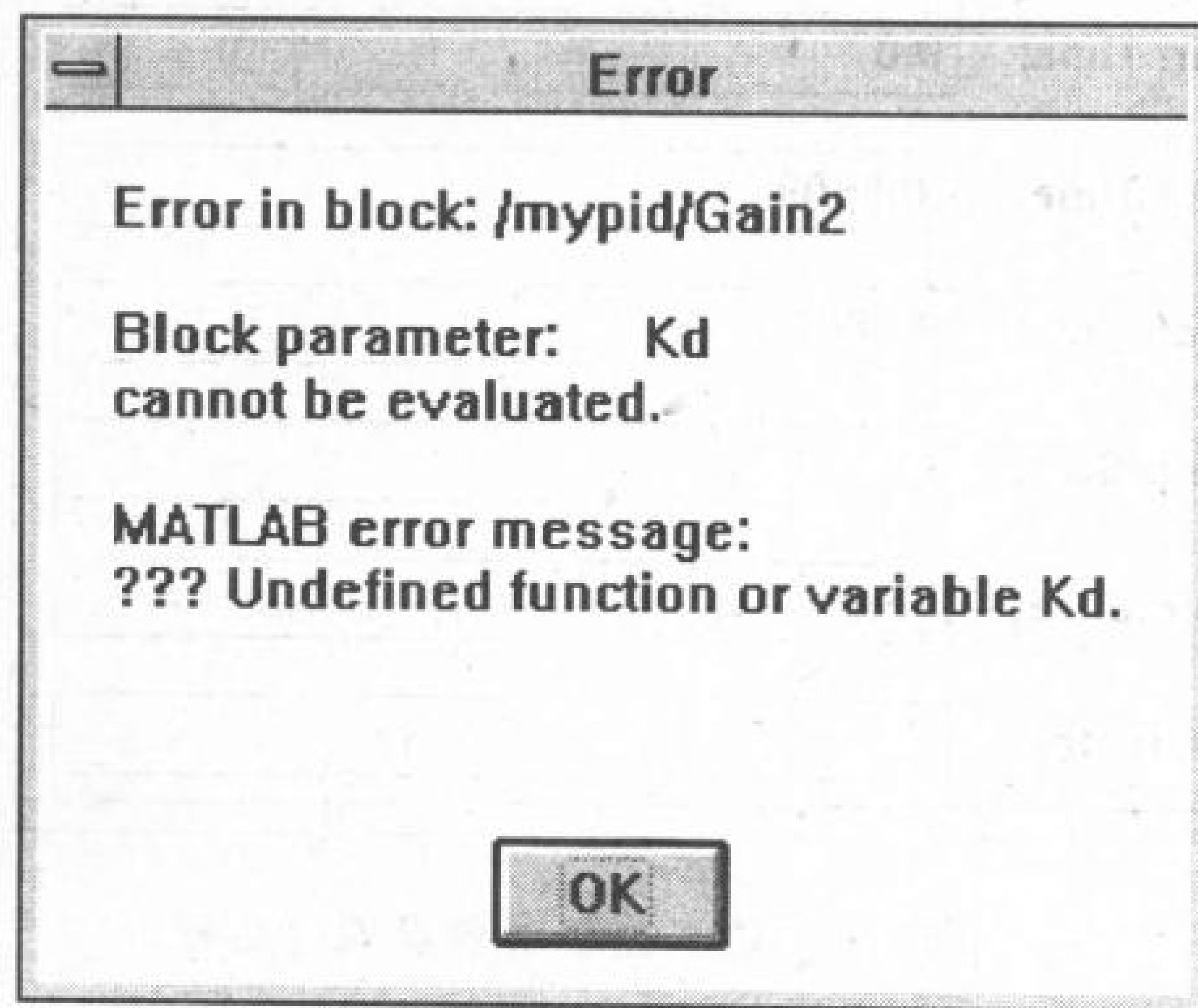


图5-24 仿真模型错误提示消息框

器作输出时，则会自动地将仿真的结果从示波器上“实时地”显示出来。例如前面的 PID 控制系统中若设置 $K_p = 10$, $K_i = 3$, $K_d = 2$, 且已知系统的模型为 $G(s) = (s^3 + 7s^2 + 24s + 24)/(s^4 + 10s^3 + 35s^2 + 50s + 24)$, 则可以得出如图 5-25 所示的示波器输出。在仿真结束时将还会自动地发出一声鸣叫，提示用户仿真过程已经完成，在一般

的应用中这一过程将十分短暂。对图 5-21 中所示的 SIMULINK 模型来说，因为时间变量和该控制系统的输出信号同时还写到 MATLAB 的工作空间变量 t 和 y 中，这样除了上面的示波器输出之外还可以采用 MATLAB 命令 plot(t,y) 将仿真结果绘制出来，这时得出的结果如图 5-26 所示，可见两种方法得出的结果是完全一致的。

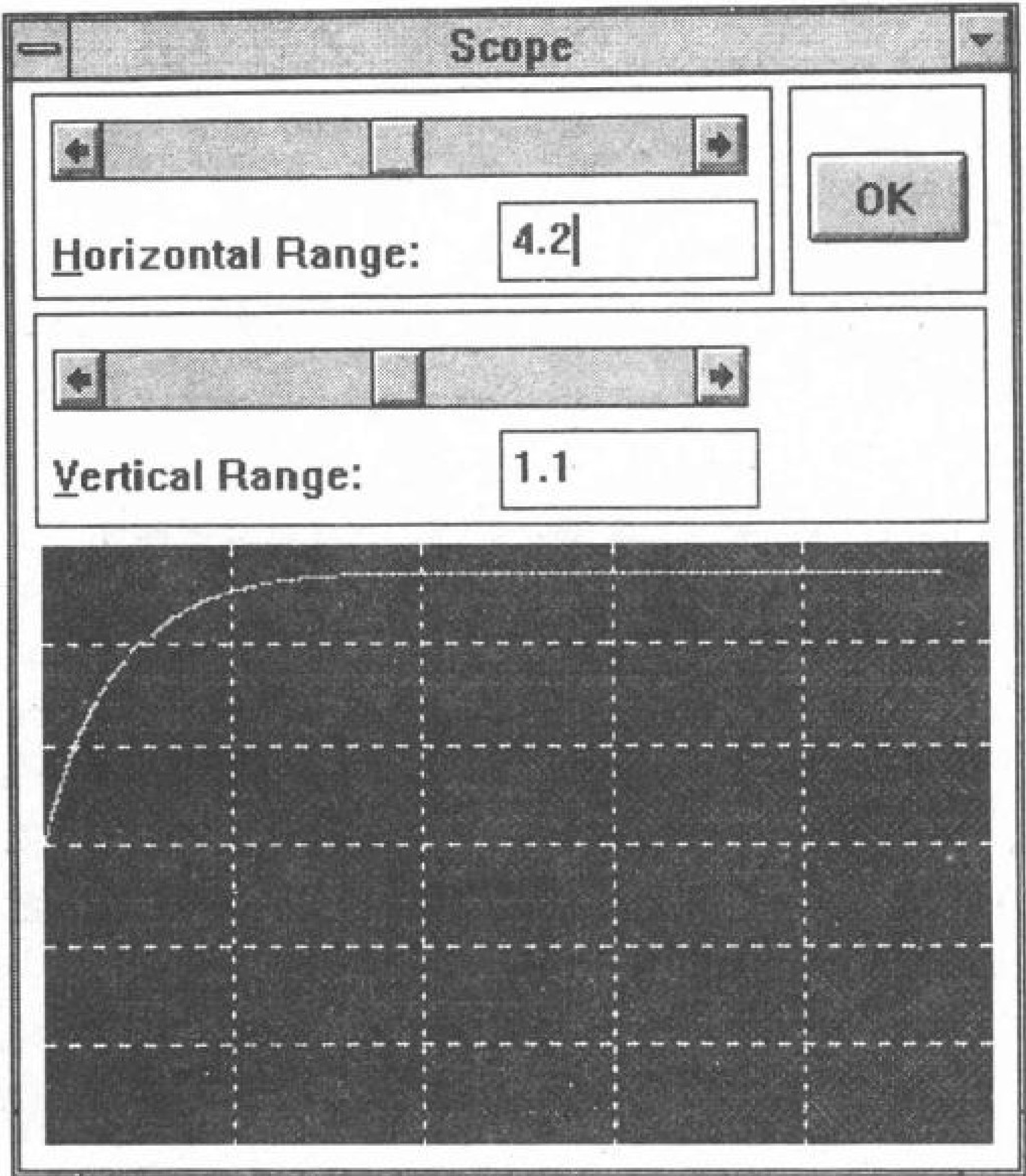


图5-25　仿真结果的示波器显示

在实际仿真过程中，用户还可以采用 Simulation 菜单下的 Pause 和 Resume 来暂停或恢复仿真过程。此外在仿真过程启动起来之后，Start 菜单项将被 Stop 菜单项取代，这时若选择 Stop 选项将中止仿真过程。

除了利用 Simulation 菜单的各个选项对仿真进行控制之外，还可以由 SIMULINK 提供的各个仿真函数来仿真给出的系统，SIMULINK 中提供的仿真函数名及意义在表 5-2 中给出。这些函数的调用格式还是相当规范的，具体表述如下

表5-2　SIMULINK 提供的仿真函数表

仿真函数名	意　　义	仿真函数名	意　　义
euler()	Euler 算法	rk23()	2/3 阶 Runge-Kutta 算法
rk45()	4/5 阶 Runge-Kutta 算法	adams()	Adams 算法
gear()	Gear 预报校正算法	linsim()	线性系统仿真算法

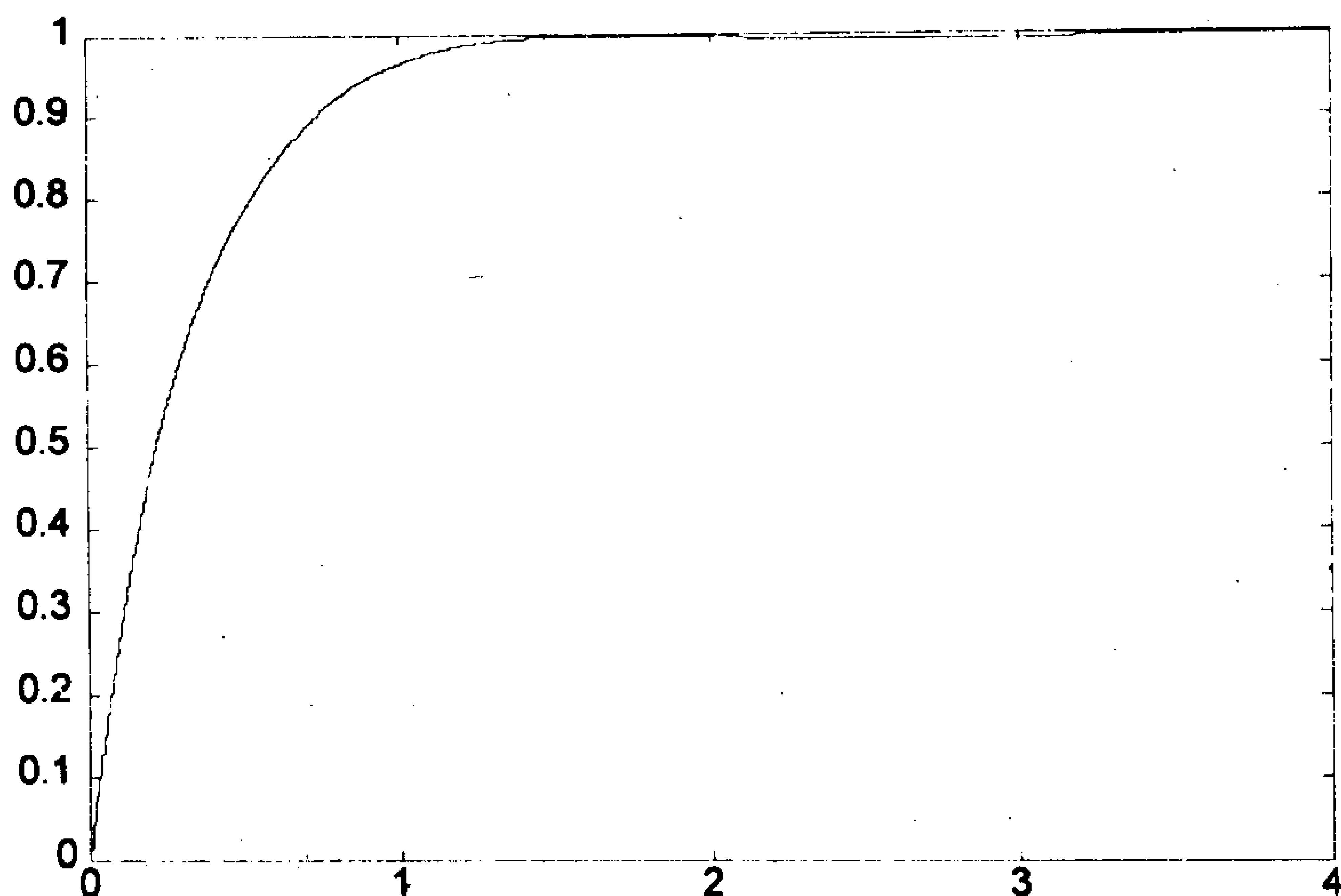


图5-26　MATLAB 绘图命令得出的系统响应曲线

[t, x, y]= 仿真函数名 (模型函数名, TF, XI, 附加选项 OPTOINS)

其中仿真函数名可以取表 5-2 中列举出的任何一个，而模型函数是由 SIMULINK 建立起来的任何 .m 文件。TF 为仿真的终止时间，它还可以写成一个 1×2 型向量 [t0, tf]，这时 t0 表示初始仿真时间，而 tf 表示终止仿真时间，XI 为状态变量的初值向量，若假定系统的初始状态变量均为 0 时，可以将此选项设置为空的矩阵 []，附加选项 OPTIONS 的取值及定义如表 5-3 所示，用户可以按对话框 5-23 的方式来调用此函数，例如，如果想对'mymodel.m' 模型用 Euler 法进行仿真，则应该给出下面的命令：

```
[t,x,y]=euler('mymodel',10,[]);
```

表5-3　附加选项 OPTIONS 表

选　项	意　义	默认值	选　项	意　义	默认值
OPTOINS(1)	误差限	10^{-3}	OPTIONS(2)	最小步长	tf/2000
OPTIONS(3)	最大步长	tf/50	OPTIONS(4)	保留选项	
OPTIONS(5)	步长范围超出时给出报警	0 (不报警)	OPTIONS(6)	是否同步绘图	0

在用函数调用的方式进行仿真之前，还需要将输出参数与 SIMULINK 的接口模块连接起来，其中 SIMULINK 的接口模块是 Connections (连接模块) 组中标注为 Output (输出模块) 的图标。

若仿真点的个数为 m，状态变量的个数为 n 而输出变量的个数为 p，则返回的各个矩阵维数如下：时间矩阵 t 为 $m\times1$ 向量，状态变量矩阵 x 为 $m\times n$ 矩阵，每一行表示在一个时刻的状态向量值，输出矩阵 y 为 $m\times p$ 矩阵，每一行表示在一个时刻的系统输出值。

5.3.3 SIMULINK 其它模块库内容概述

SIMULINK 提供了比较丰富的模块库，一般在控制系统的分析与设计中遇到的模块都可以从模块库中找到，比如离散时间系统的环节可以由 Discrete 模块库中找出，见图 5-27(a) 所示，而非线性环节的模块可以由 Nonlinear 模块库中找出，见图 5-27(b)，一

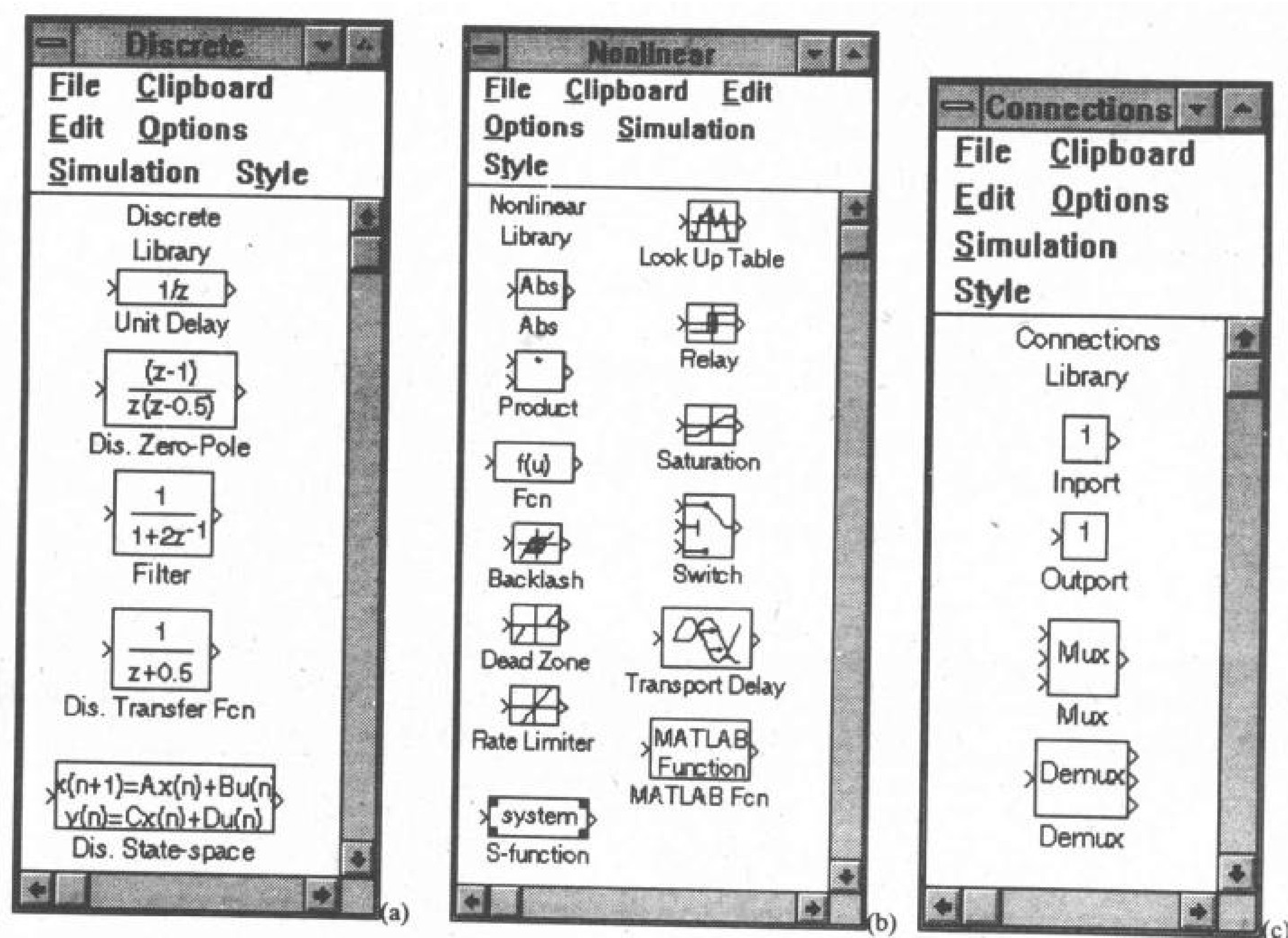


图5-27 SIMULINK 提供的其它模块库

些接口类模块置于 Connections 模块组中，双点该图标可见图 5-27(c)。这些环节的参数也是容易调节的，比如饱和非线性环节 (Saturation) 的参数设置可以通过双点该图标来完成，这时将得出一个如图 5-28 所示的对话框，用户只需在该对话框中 Lower output limit (输出下限) 和 Upper output limit (输出上限) 引导的两个编辑框中填写上合适的数据就可以了。

除了上面列出的各个基本模块库之外，SIMULINK 的每一个新的版本都将加入一些新的模块库，如果用户用鼠标双点图 5-10 中的 Extras 图标，则将获得如图 5-29 所示的附加模块库表示。

可以看出在该模块库中定义了很多其它的模块，包括附加的输入函数模块库，如图 5-30 (a) 所示，附加离散时间模块库，如图 5-30(b) 所示。用户还可以根据需要自己对模块库进行扩充，形成自己的模块库，具体的扩充方法将在下节中介绍。

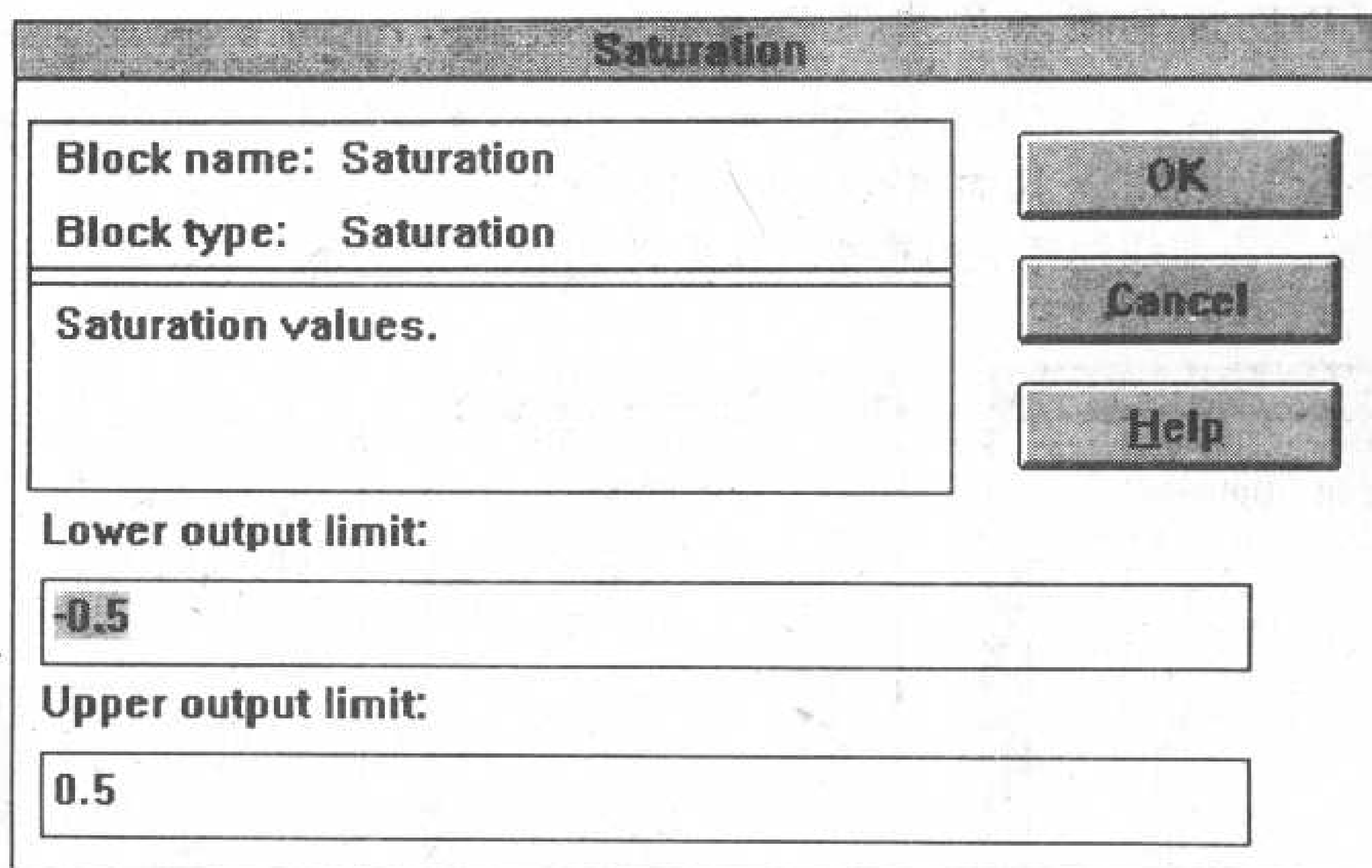


图5-28 饱和非线性环节参数修正对话框

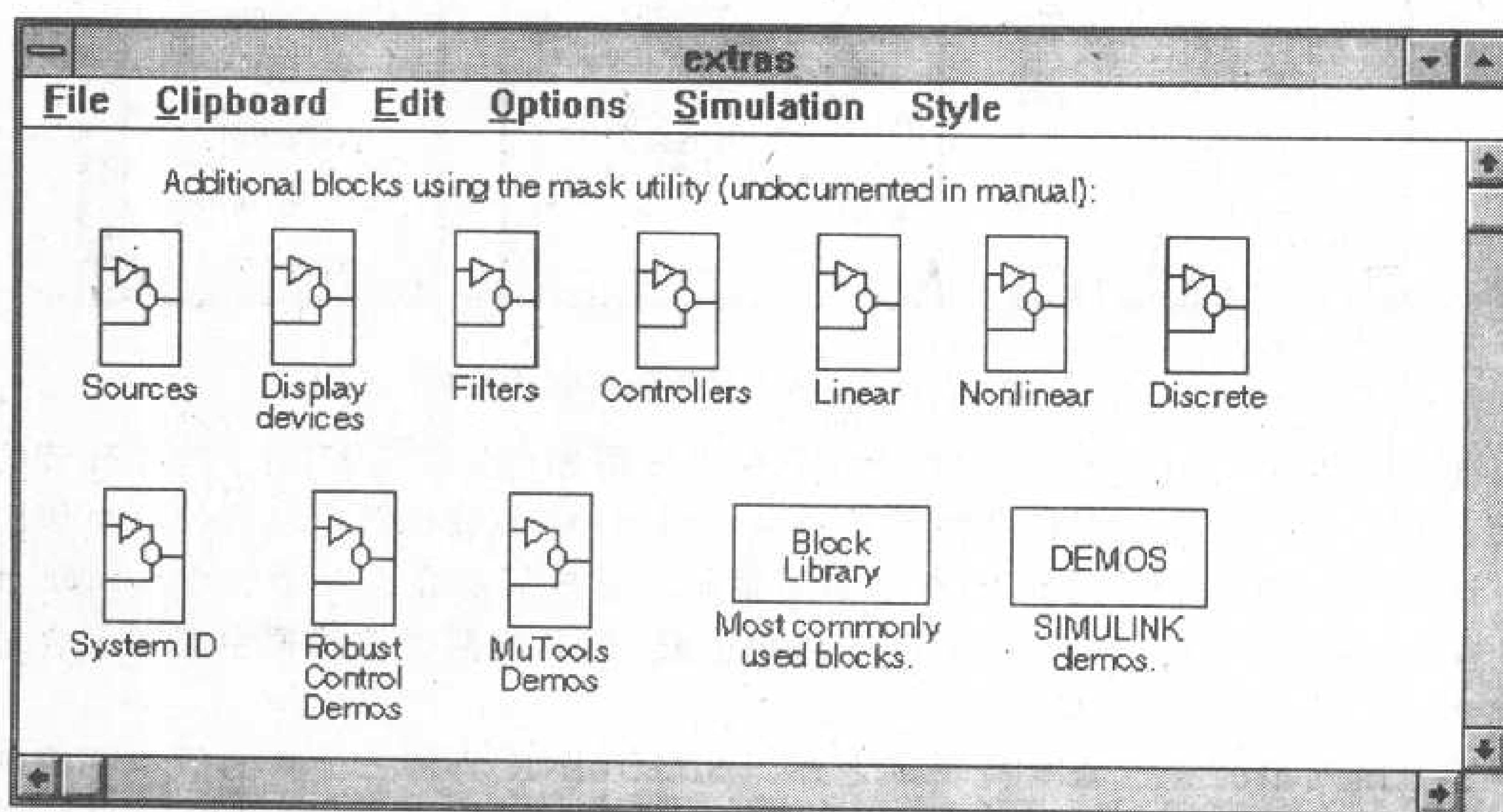


图5-29 附加模块库的内容

5.4 SIMULINK 使用的高级技术

从前面的介绍中可以看出利用 SIMULINK 软件可以容易地构造出系统的结构图模型，对于一些公共的部分还可以把它单独提出来构成一些实用的新模块以备以后使用，例如前面例子中的 PID 控制器就可以单独提取出来构成一个在 SIMULINK 下直接可以使

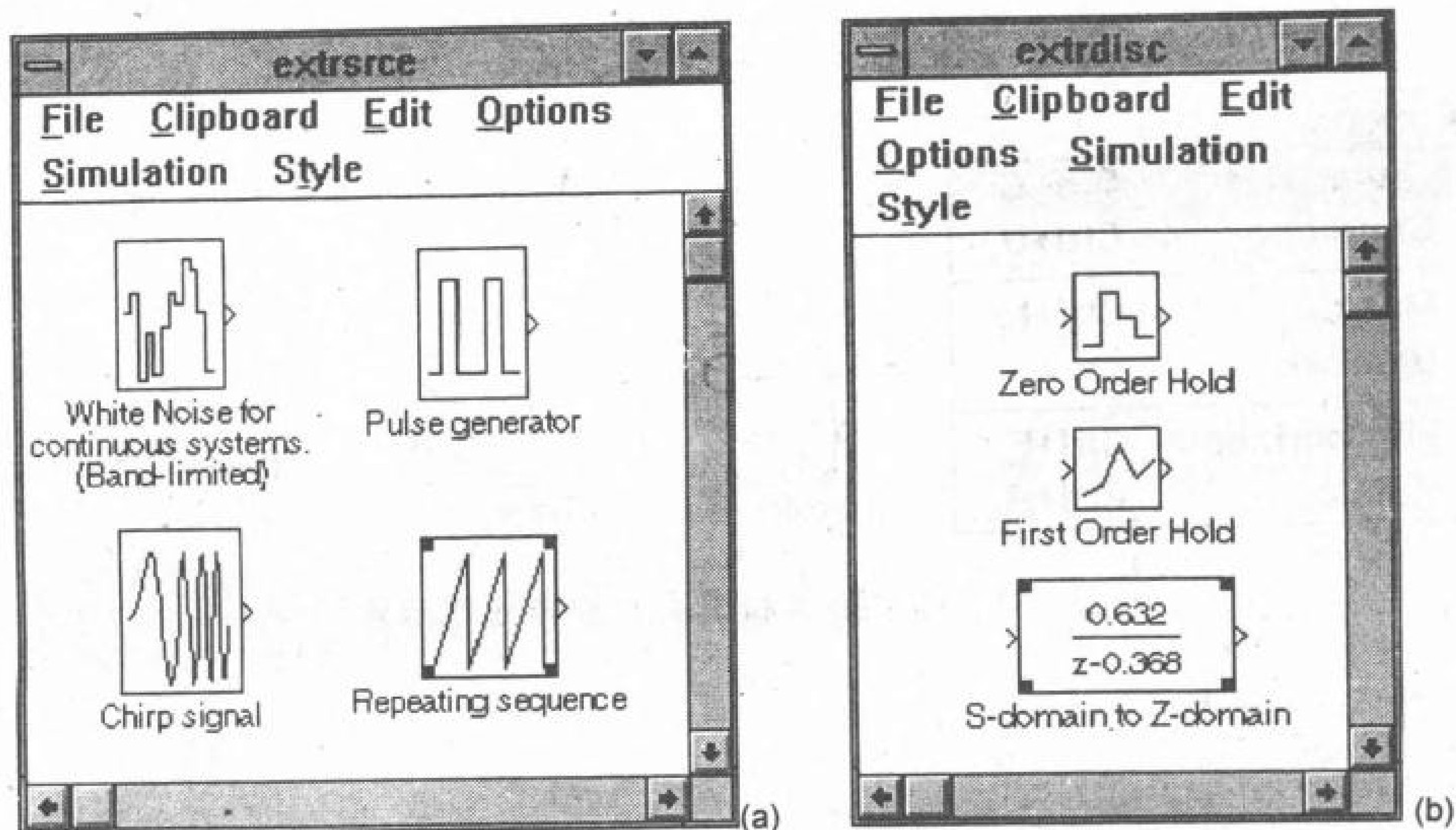


图5-30 附加输入与离散时间模块库

用的模块[1]，用户既可以构成一个模块组，也可以通过封装的方法构造一个 SIMULINK 型的子模块，允许用户按照图 5-28 的格式输入有关的参数。

5.4.1 SIMULINK 模块的安排

在绘制系统框图时经常会有这样的要求，即有些模块是否能较方便地翻转或旋转一下，使得得出的框图更优美更清晰，比如说，如果在反馈控制回路中有一个线性传递函数环节，这样就往往希望对标准的传递函数模块 (默认状态是输入端在左边而输出端在右边) 作 180° 的翻转，使得其输入端在右边而输出端在左边，这样会使得连线更容易，并可以避免不必要的交叉线，使得框图读起来更清晰。在 SIMULINK 下实现这一功能是轻而易举的事情，首先用鼠标点中的方法选中要旋转处理的模块，然后打开 Options 菜单 (如图 5-31(a) 所示)， 从中选择 Flip Horizontal 就可以将此模块作 180° 的翻转，即可得到如图 5-31(b) 所示的效果， Options 菜单中的另一个选项 Rotate 会每次将选中的模块旋转 90°，则得出如图 5-31(c) 所示的效果。用户可以通过这样的方法来对选择的模块作简单地安排，使之看起来更美观。

SIMULINK 还允许将若干个模块用一个模块组的形式来表示，首先可以将要构成模块组的所有子模块同时选中[1]，例如图 5-32(a) 中给出的 PID 控制器的所有模块处于选中的状态， 然后选择 Options | Group 菜单项，则会自动地将这些子模块构成一个模块组的标志，如图 5-32 (b) 所示，这样此模块以后就可以拿来作为一个公用的模块来使用了。如果用户想改变模块组中的具体内容，则需用鼠标左键双点该组的图标，这时就会

[1]事实上 SIMULINK 已经给出了各种 PID 控制模块，这里只把它作为例子来演示如何构造新模块。

[1]选中的方法是用鼠标器在各个子模块的周围画出一个矩形框，具体的方法是在矩形的左上角处点下鼠标左键，然后拖动鼠标器直至右下角处释放鼠标键，这时将在各个模块上用黑点标出以表示已经选中。

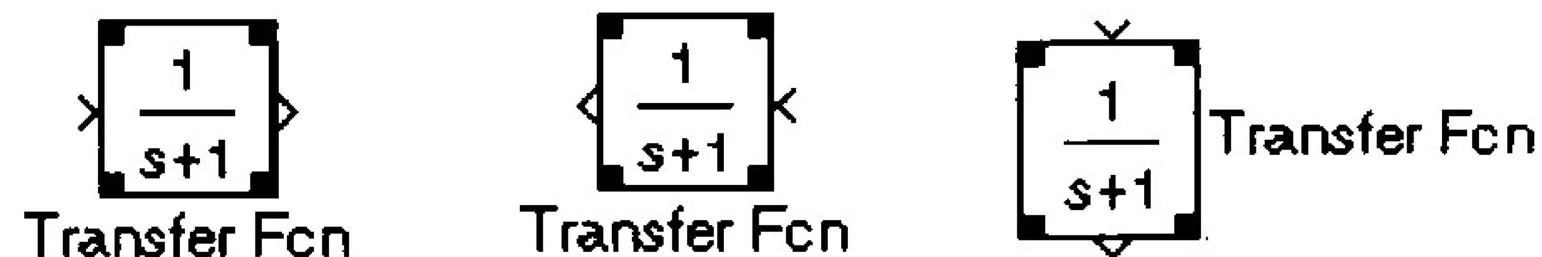


图5-31　选项菜单和模块的旋转处理示意

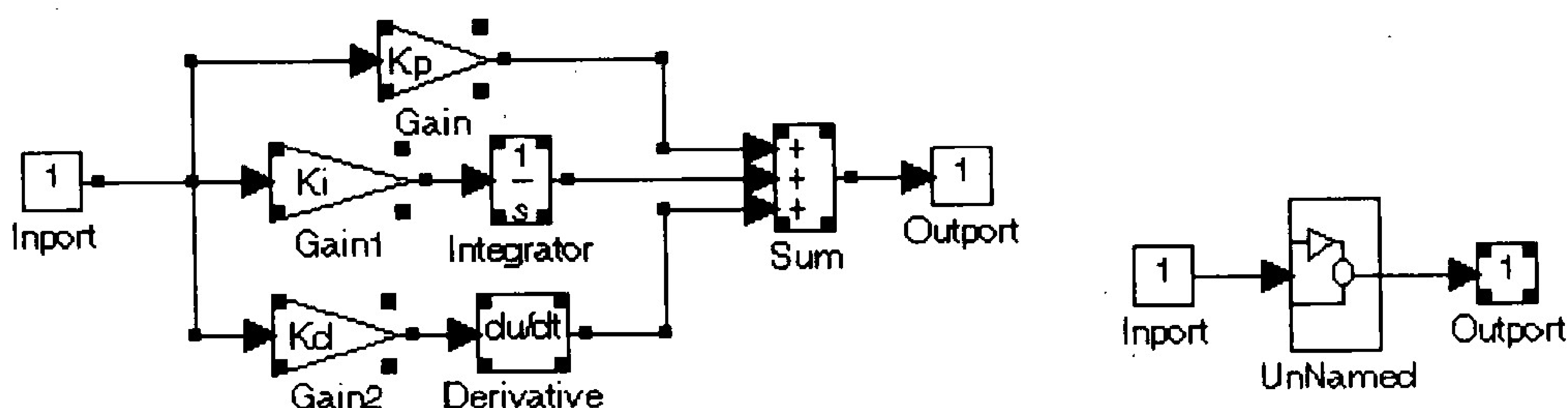


图5-32　模块的 Group 处理

自动地弹出一个子模型窗口，将该模块的具体内容显示出来，用户可以在这一窗口内修改该模块的内容。用户还可以容易地修改各个图标的说明文字，这需要首先用鼠标选择要修改的文字，然后用键盘的方式将文字修改成所需要的内容，例如首先可以点选新构造的图标下的 NoName 说明文字，然后拖动鼠标，直至整个说明文字变成蓝色 (即说明该文字被选中)，再用键盘输入 PID Controller 字样，这时将把该图标的说明文字改变成 PID Controller，如图 5-32(c) 所示。

如果用户想消除原有的模块组，而想返回到原来的单独模块表示形式，则需要首先选中该模块组的图标，然后选择 Options | Ungroup 菜单项，这时就会自动地将原来的模块组表示形式消除，恢复到构成组之前的表示。

5.4.2　构造 SIMULINK 型模块

按照前面的方法构成模块组之后，有时往往因为修改起来较困难而觉得模块组的用法不是很方便，这时需要将该模块组的具体内容封装起来，使得用户不能看到模块组的具体内容，只可以对模块组允许的接口数据进行修改，这就需要使用 Options | Mask (封装模块) 选项了，首先构造一个模块组，然后用鼠标左键点取的方法选择它，在选择 Options | Mask 菜单项，这时将给出一个如图 5-33 所示的对话框，用户可以在该对话框中给出各种封装命令，在此对话框中需要对如下的一些编辑框作出说明。

UnNamed

Block name: UnNamed

Block type: UnNamed

Mask Block Definitions

OK

Cancel

Help

New block type:

PID Controller

Dialog strings separated by | :

PID Controller \n u=Kp u + Ki (Integral u) + Kd du/dt |Pr

Initialization commands:

Kp=@1; Ki=@2; Kd=@3

Drawing commands:

PID

Help string:

图5-33 模块封装对话框

- **新模块标识** (new block type): 用户可以写入任何字符串来作为标识。
- **对话框命令** (dialog strings): 用户可以在其中填写封装后对话框的提示参数，这些参数用 | 符号来分隔，其中前一个 | 符号前的部分作为简要提示给出，其作用由下面的 PID 封装模块命令中可以立即看出

```
PID Controller \n    u=Kp u + Ki (Integral u) + Kd du/dt |
    Proportional Kp: | Integral Ki | Derivative Kd
```

键入封装命令之后，该模块就会自动封装起来了，以后若想用鼠标左键双点的方法打开该对话框，就会得出如图 5-34 所示的对话框， 这时该窗口中的提示文字是由下面的命令所设定的，其中 \n 和 C 语言的记号一样表示换行显示， | 标志表示换项。如果在这里不给出对话框命令，而只给出 MATLAB 命令，则封装之后再双点该模块时会自动地执行这里的 MATLAB 命令。例如在 F-14 演示程序中出现的 Load Data 图标就是由下面的命令而表示的： eval('f14dat') 。

PID Controller (Mask)

Block name: PID Controller

Block type: PID Controller (Mask)

PID Controller
u=Kp u + Ki (Integral u) + Kd du/dt

OK

Cancel

Help

Proportional Kp:

Integral Ki

Derivative Kd

图5-34 封装后参数输入对话框

- **初始化命令** (initialization commands): 用来和对话框中提供的数据建立联系，例如在对话框中给出 PID 控制器的 3 个参数 K_p, K_i 和 K_d, 则可以通过下面的命令加以赋值

```
Kp=@1; Ki=@2; Kd=@3;
```

其中 @ 符号表示对话框中参数代号，例如 @1 表示取对话框中的第一个参数，即图 5-34 中所示的 Kp 参数。这些参数还可以由 MATLAB 加以重新运算，例如若系统的 PID 参数由下式定义

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + K_d s\right), \quad 亦即\ T_i = \frac{K_p}{K_i},\ T_d = \frac{K_p}{K_d}$$

这样应给出下面的命令 `Kp=@1; Ti=@1/@2; Td=@1/@3;`

- **绘图命令** (drawing commands): 在这里可以给出画图命令来修饰产生的图标的内部表示方法，可以使用 `plot()` 等函数来修饰。
- **帮助字符串** (help string): 给出关于此模块的进一步帮助信息，在点中封装后模块对话框中的 Help 按钮时，将显示进一步的帮助信息。

5.4.3 模型线性化方法及 SIMULINK 实现

若可以由一阶微分方程组 (亦即状态方程) 的形式写出系统的模型

$$\dot{x}_i(t) = f_i(x_1, x_2, \cdots, x_n, u, t) \tag{5.4.1}$$

其中 $u(t)$ 和 t 分别为系统的输入信号和时间，且 $x_i(t), i=1,\cdots,n$ 为系统的状态变量，而 $f_i(\cdot)$ 为给定的函数，则系统的平衡点可以由假设 $\dot{x}_i(t)=0$ 的方式来求出，因为此时由于状态变量的变化率为零，可以认为系统处于静态。

也可以由 SIMULINK 模型来求出系统的平衡点，在绘制 SIMULINK 模型时注意首先应该将系统的输入和输出用 Connections 模块组中的 Import 和 Outport 模块的接口形式来表示，这样就可以由 SIMULINK 提供的 trim() 函数来求出系统的平衡点，该函数的调用格式如下：

```
[x,u,y,dx]=trim(模型名,  x0,u0,y0)
```

其中模型名即为模型的 SIMULINK 表现形式，x0，u0，y0 分别为系统的状态向量，输入和输出向量的初始值，通过此函数的调用将返回系统在平衡点处的状态向量 x，输入向量 u 和输出向量 y 的值，并给出状态向量的变化率 dx（一般为很小的数值），该函数是通过极小化的算法来求出系统的平衡点的，所以有时不能保证状态向量的变化率等于零，只能使之趋于零。在函数调用时其中的一些参数可以省略，例如该函数可以简单地写成

```
[x,u,y,dx]=trim(模型名)
```

这时会在默认的输入与输出下求出系统的平衡点来，这样的方法尤其对线性系统是有效的。

例 5.8　假设第 4 章中介绍的 F-14 战斗机模型的 SIMULINK 文件名为 'f14.m'(该文件由 SIMULINK 的演示程序提供)，则首先可以在 MATLAB 环境下键入 f14 命令，再从得出的模型窗口中双点 Load Data 按钮，这时就可以通过下面的命令来求出系统的平衡点

```
>>  [x,u,y,dx]=trim('f14');
>>  x'
ans =  0.0015   -0.0040    0.0002    0.0000    6.4252    0.5649    0.0000
      -0.0010   -0.0012   -0.0002
>>  u
ans = -1.0023
>>  y'
ans =  0.0000   -0.0858
>>  dx'
ans = 1.0e-014 *
      -0.8276   -0.0089    0.0001    0.0369    0.0000   -0.6256    0.0000
       0.0000         0    0.0022
```

可见得出的状态向量的变化率属于 10^{-14} 量级，所以还是很小的。

系统的线性化实际上是求取非线性系统在某工作点处近似的线性系统模型的过程，对式 (5.4.1) 中给出的非线性系统状态方程模型来说，可以在工作点 x_0，u_0 处作下面的近似

$$\Delta\dot{x}_i=\sum_{j=1}^{n}\left.\frac{\partial f_i(x,u)}{\partial x_j}\right|_{x_0,u_0}\Delta x_j+\sum_{j=1}^{p}\left.\frac{\partial f_i(x,u)}{\partial u_j}\right|_{x_0,u_0}\Delta u_j \tag{5.4.2}$$

这样构造新的系统状态变量，则可以得出下面的线性化模型

$$\Delta\dot{x}(t) = A_l\Delta x(t) + B_l\Delta u(t) \tag{5.4.3}$$

其中

$$A_l = \begin{bmatrix} \partial f_1/\partial x_1 & \cdots & \partial f_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \cdots & \partial f_n/\partial x_n \end{bmatrix}, \quad B_l = \begin{bmatrix} \partial f_1/\partial r_1 & \cdots & \partial f_1/\partial r_p \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial r_1 & \cdots & \partial f_n/\partial r_p \end{bmatrix} \tag{5.4.4}$$

SIMULINK 也提供了对 SIMULINK 模型进行线性化的函数 linmod2()，该函数的调用格式为

```
[A,B,C,D]=linmod2(模型名,  x,u)
```

这里 x 和 u 分别为平衡点处的状态向量和输入向量，通过该函数的调用将得出线性化模型的状态方程参数 (A,B,C,D)。对线性系统来说，平衡点参数 x 和 u 是可以省略的，故上面的函数调用格式可以简化成

```
[A,B,C,D]=linmod2(模型名)
```

对上面介绍的 F-14 系统进行线性化可以采用下面的命令

```
>>  [A, B, C, D]=linmod2('f14');
```

其结果已经在第 4 章中给出，这里就不再重复了。由此线性化模型也可以对系统进行仿真分析，例如可以由下面命令得出系统的输出阶跃响应曲线，如图 5-35 所示。

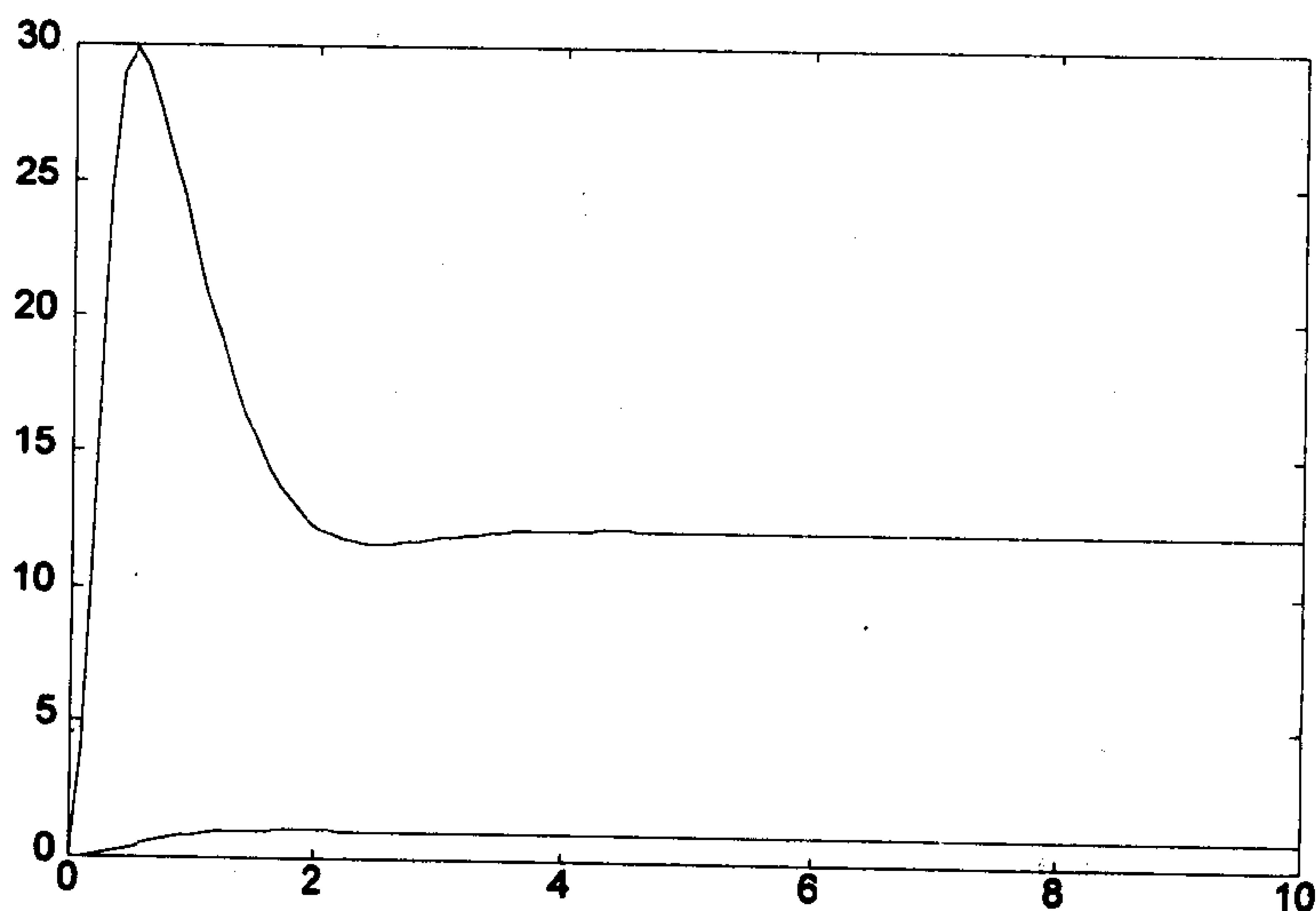


图5-35 F-14 系统的阶跃响应曲线

```
>> t=0:.1:10; y=step(A, B, C, D, 1, t);
>> plot(t,y)
```

5.4.4 S 函数的编写与使用

用户建立起 SIMULINK 系统模型时就会建立一个相应的 S 函数，比如考虑第 1 章中给出的 Van der Pol 方程，如果其状态方程可以写成

$$\dot{y}_1 = y_1(1 - y_2^2) - y_2,\quad \dot{y}_2 = y_1$$

则可以建立起如图 5-36 所示的 SIMULINK 模型结构，该模型可以直接生成下面的 S 函数

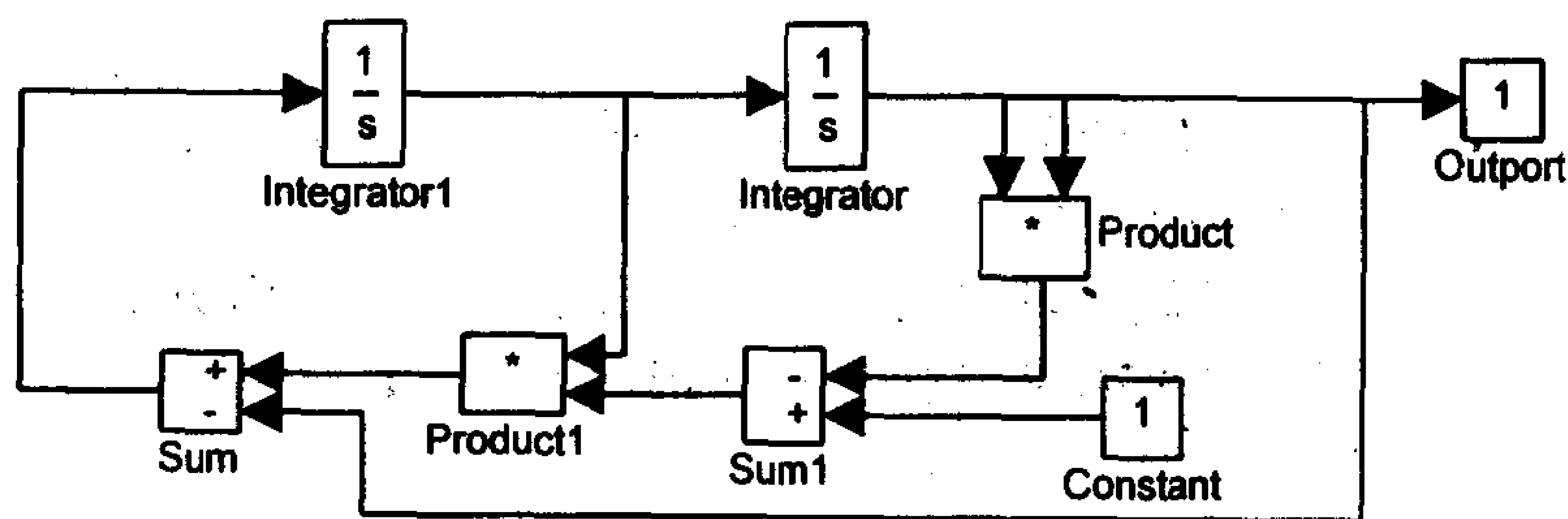


图5-36 Van der Pol 方程的 SIMULINK 描述

```
function [ret,x0,str]=vdps(t,x,u,flag);
sys = mfilename;  new_system(sys);  simver(1.2)
if(0 == (nargin + nargout)), set_param(sys,'Location',[100,100,600,400]);
open_system(sys); end;
set_param(sys,'algorithm',  'RK-45'); set_param(sys,'Start time', '0.0')
set_param(sys,'Stop time', '999999'); set_param(sys,'Min step size', '0.0001')
set_param(sys,'Max step size', '10'); set_param(sys,'Relative error','1e-3');
set_param(sys,'Return vars', '');
add_block('built-in/Integrator',[sys,'/','Integrator'])
set_param([sys,'/','Integrator'],'Initial','0.25','position',[200,22,220,58]);
add_block('built-in/Product',[sys,'/','Product'])
set_param([sys,'/','Product'],'orientation',1,'position',[244,65,271,85])
add_block('built-in/Sum',[sys,'/','Sum1'])
set_param([sys,'/','Sum1'],'orientation',2,'inputs','-+','position',[210,103,230,127])
add_block('built-in/Constant',[sys,'/','Constant'])
set_param([sys,'/','Constant'],'orientation',2,'position',[275,110,295,130])
add_block('built-in/Product',[sys,'/','Product1'])
set_param([sys,'/','Product1'],'orientation',2,'position',[140,100,165,120])
add_block('built-in/Sum',[sys,'/','Sum'])
set_param([sys,'/','Sum'],'orientation',2,'inputs','+-','position',[65,105,85,125])
add_block('built-in/Integrator',[sys,'/','Integrator1'])
set_param([sys,'/','Integrator1'],'Initial','0.25','position',[105,22,125,58])
add_block('built-in/Outport',[sys,'/','Outport'])
```

```
set_param([sys,'/','Outport'],'position',[330,30,350,50])
add_line(sys,[130,40;190,40]);  add_line(sys,[225,40;320,40])
add_line(sys,[225,40;250,40;250,55]); add_line(sys,[260,90;260,110;240,110])
add_line(sys,[270,120;240,120]); add_line(sys,[130,40;180,40;180,105;175,105])
add_line(sys,[205,115;175,115]); add_line(sys,[135,110;95,110])
add_line(sys,[225,40;315,40;315,145;115,145;115,120;95,120])
add_line(sys,[60,115;30,115;30,40;95,40]); add_line(sys,[225,40;265,40;265,55])
if (nargin | nargout), if (nargin > 3)
   if (flag == 0), eval(['[ret,x0,str]=',sys,'(t,x,u,flag);'])
   else, eval(['ret =', sys,'(t,x,u,flag);']); end
else,      [ret,x0,str] = feval(sys); end; end
```

可见，该函数中大量调用了 add_line(), add_block() 和 set_param() 函数，这些函数是 SIMULINK 环境所提供的，其调用方法是很显然的。这一函数除了用来对原始模型进行描述以外，还可以绘制出系统的框图结构，所以程序显得很繁琐。若用户不想再绘制出系统的结构图，而只想对系统进行仿真分析，则有必要按照规则建立某种简单的描述方法。

SIMULINK 提供了书写简单 S 函数的方法，其引导语句格式为

```
function [sys, x0]=model(t, x, u, flag)
```

其中 model() 为模型函数的函数名，例如前面的函数中使用了 vdps() 作为该函数的函数名，t，x，u 为对应于状态方程模型的时间、状态向量和输入向量，flag 为选项位，用于标识该函数的返回结果，例如 flag 为 1 时，变量 sys 将返回系统的状态向量，flag 为 0 时 sys 和 x0 将分别返回系统的阶次信息变量和初始状态，而 flag 为 2 时 sys 变量将返回系统的输出向量。

例如同样的 Van der Pol 方程可以用 S 函数描述为

```
function [sys, x0]=new_vdp(t, x, u, flag)
if flag==0, sys=[2;0;0;0;0;0]; x0=[0.25; 0.25];
elseif abs(flag)==1, sys=[x(1)*(1-x(2)*x(2))-x(2); x(1)];
else, sys=[]; end
```

可见用这样的方法可以简单地建立起系统的 S 函数模型，以后此模型也可以直接使用和 vdps() 函数同样的格式来进行调用，这可以大大地提高仿真的效率。为了进一步提高仿真的效率，可以将上面的 S 函数用 C 或 FORTRAN 语言来进行改写，这样的 C 或 FORTRAN 语言程序可以由标准的格式来编写，然后再由 MATLAB 提供的功能转换成 MEX 文件。前面的 Van der Pol 方程可以由 C 语言写出如下的程序

```
#define NSTATES 2
void init_conditions(x0)
double *x0;
```

```
{     x0[0]=0.25; x0[1]=0.25;     }
void derivatives(t,u,x,dx)
double t, *x, *u, *dx;
{     dx[0]=x[0]*(1-x[1]*x[1])-x[1]; dx[1]=x[0];     }
#include "simulink.h"
```

熟悉 C 语言编程的读者可以阅读一下所包含的 simulink.h 头文件的内容。

写出了模型的 S 函数之后，就可以使用下面的 MATLAB 命令来绘制 Van der Pol 方程系统的时间响应图和相平面图形，如图 5-37 (a) 和 (b) 所示。

```
>>   [t,x]=rk45('new_vdp',20);
>>   subplot(121); plot(t,x)
>>   subplot(122); plot(x(:,1),x(:,2))
```

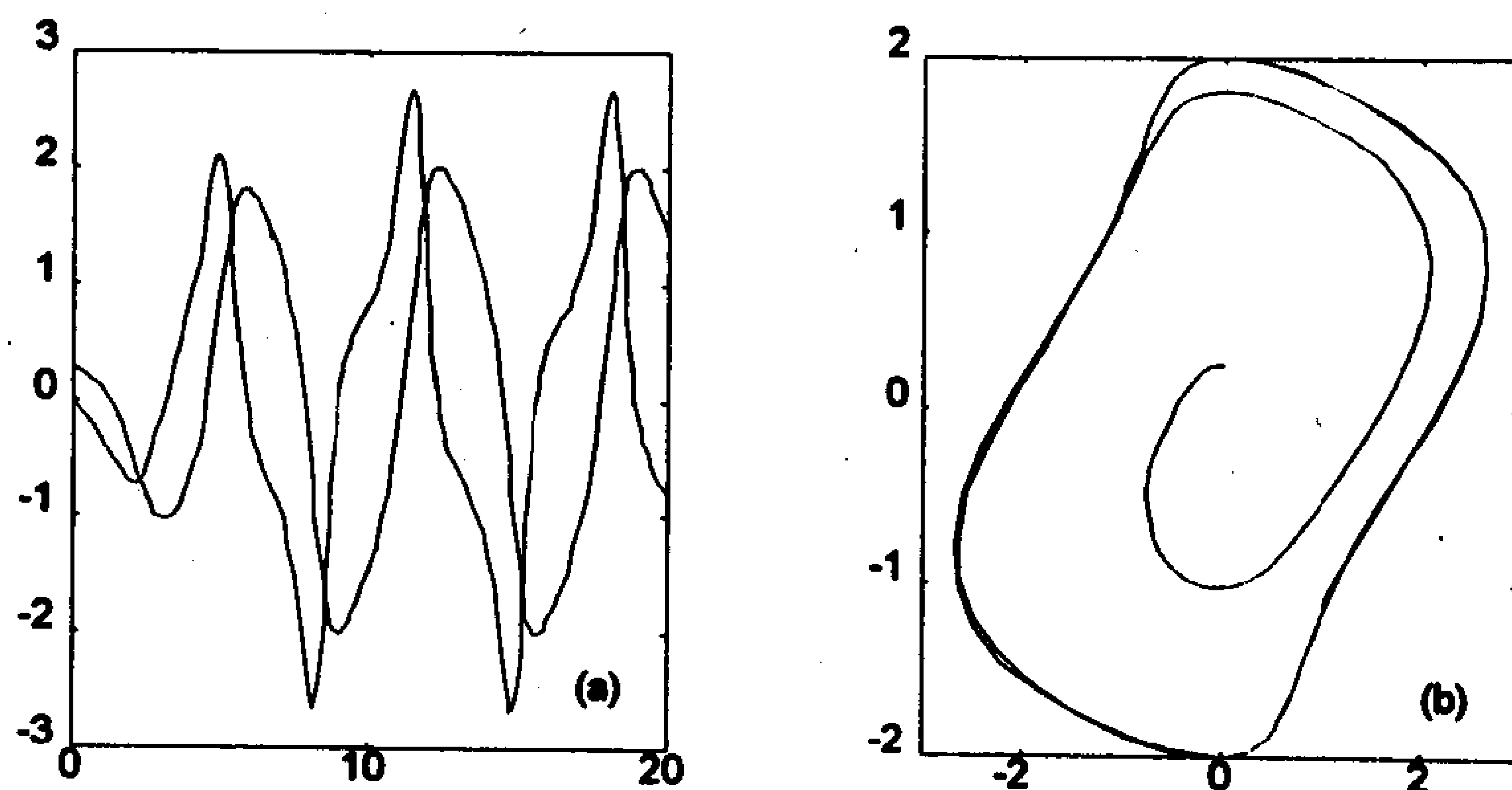


图 5-37　Van der Pol 方程的时间响应与相平面图形

5.4.5　SIMULINK 界面设置

SIMULINK 提供了对界面美观性进行设计的实用方法，例如它允许用户比较容易地设置模型界面的颜色、字体等，甚至可以在选定的方框上加阴影类修饰。这就要求用户首先打开 SIMULINK 界面上 Style 菜单，如图 5-38 (a) 所示，例如用户可以选择其中的 Background Color (背景颜色), Foreground Color (前景颜色) 或 Screen Color (屏幕颜色) 等选项将得出图 5-38 (b) 所示的下级子菜单，从中可以任意地选择颜色，给所选择的对象加以颜色修饰，这无疑将使得得出的 SIMULINK 模型显得美观。用户还可以选择 Drop Shadow (加阴影) 菜单选项给选中的对象加阴影的修饰。另外用户还可以选择 Font 菜单项对选中的对象作字体上的修饰。

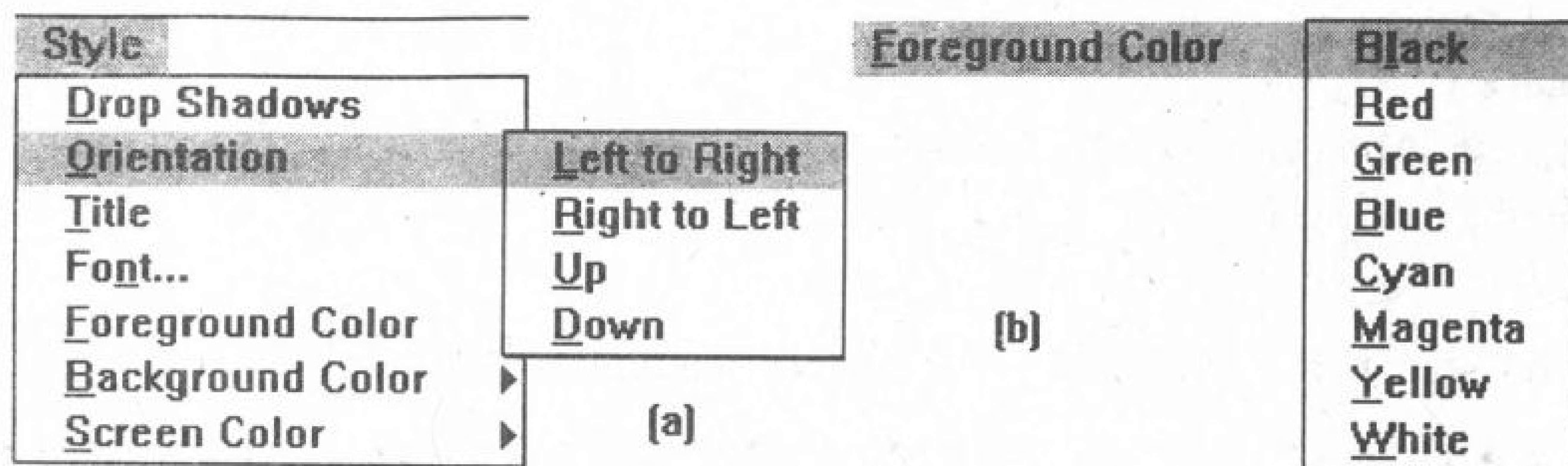


图5-38 SIMULINK 程序的 Style 菜单

5.5 SIMULINK 仿真举例

由前面的叙述中可以看出 SIMULINK 是一个功能十分强大的仿真软件，其特点主要表现在系统框图建立十分容易并直观，此外和 MATLAB 一样， SIMULINK 的仿真精度是比较高的，所以得出的仿真结果是可信的。本节中将通过一些具体的实例来演示 SIMULINK 的仿真方法及结果。

例 5.9 首先考虑 F-14 战斗机系统的仿真分析问题，在第 4 章中曾经引入了 F-14 战斗机的数学模型问题，并给出了 SIMULINK 提供的模型表示。该演示是 SIMULINK 提供的一个演示性模型，用户只需在打开 SIMULINK 中的 Extras 模块，再选择其中的 SIMULINK Demos 模块，就可以获得一组新的演示选项，然后从中选择 F-14 Flight Control 模块，这样就可以获得第 4 章中的 F-14 系统的 SIMULINK 表示，可见，这一系统被分割成了若干个具有互连关系的子系统，例如其中的 Controller 子系统是由图 5-39 所示的 SIMULINK 模型构成的，每个子模块可以通过双点该图标的方法来打开，并可以进行随意的修正。

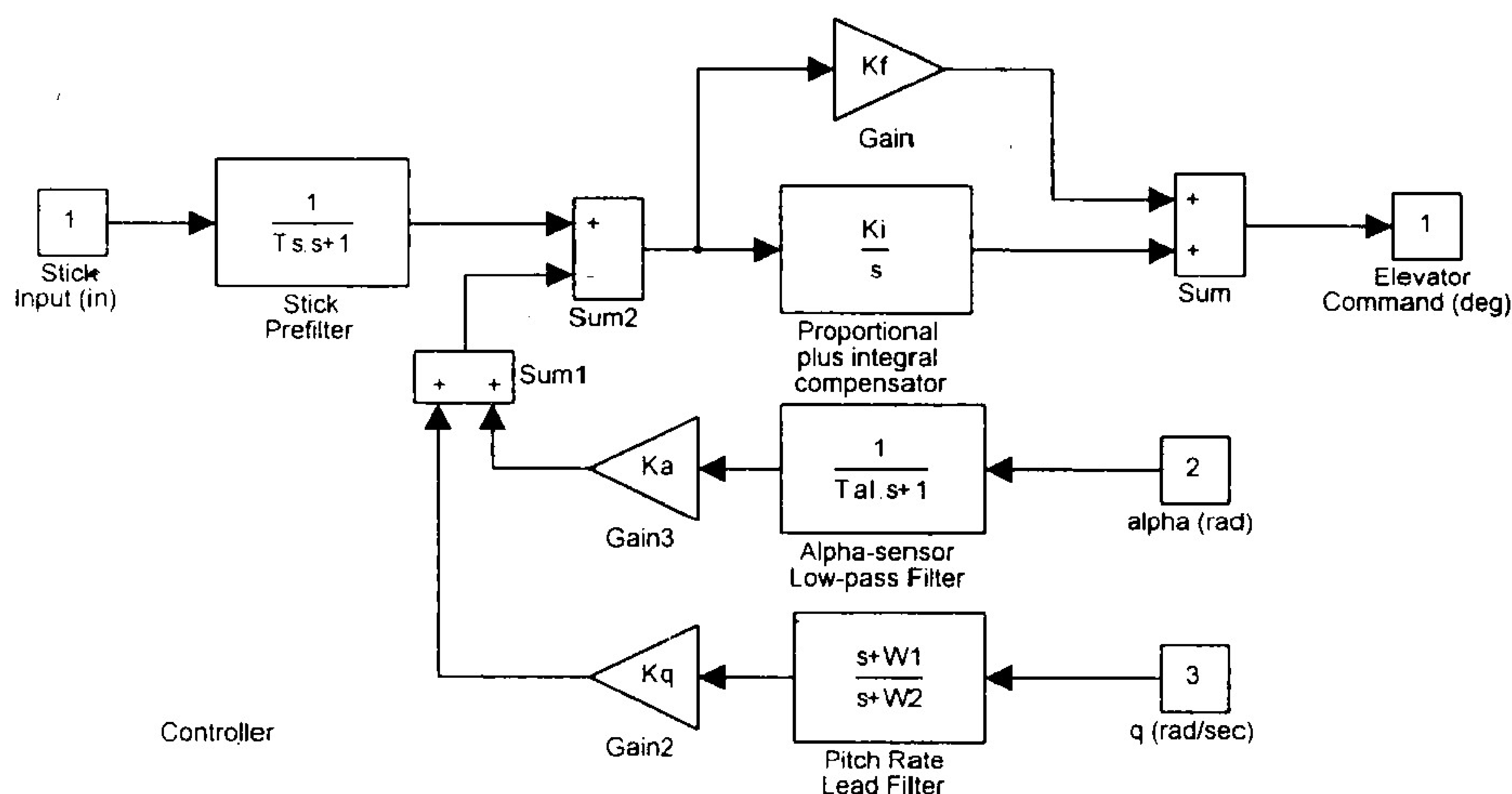


图5-39 F-14 模型中控制器的 SIMULINK 子模型

将一个复杂的大系统划分为若干个子系统的方法在实际应用中是经常使用的，利用这样的方法可以以简洁的形式将系统的模型表示出来。

在执行此程序之前，首先应该读入数据，这是由双点该框图中的 Load Data (读入数据) 按钮所激活的，输入数据之后，就可以对原始系统进行仿真了，比如说若想获得 Pilot Acceleration 信号，则可以打开标注为 Pilot G force scope 的示波器，并选择 Simulation | Start 菜单项来启动仿真过程，这样就可以"实时地"获得如图 5-40 所示的响应曲线示波器表示。

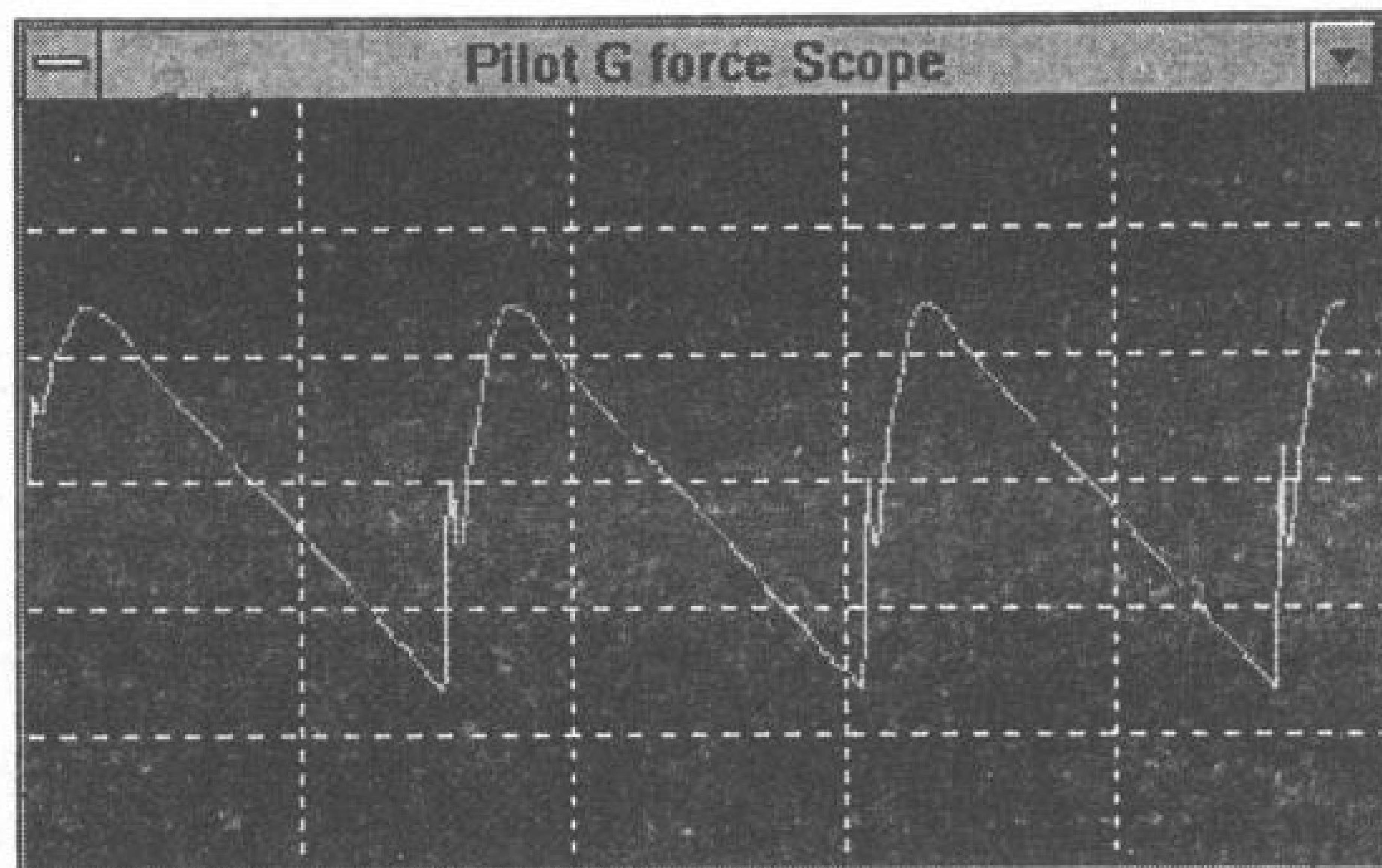


图5-40 Pilot Acceleration 示波器显示

例 5.10 考虑模型参考自适应系统的一个简单例子，即对 2 阶系统的增益进行自适应调节，假设 2 阶系统的模型为

$$G(s)=\frac{b_0}{a_2s^2+a_1s+1}$$

由超稳定性设计理论可以构造出如图 5-41 所示的控制结构 [10]，在该框图中运用了两个乘法器来

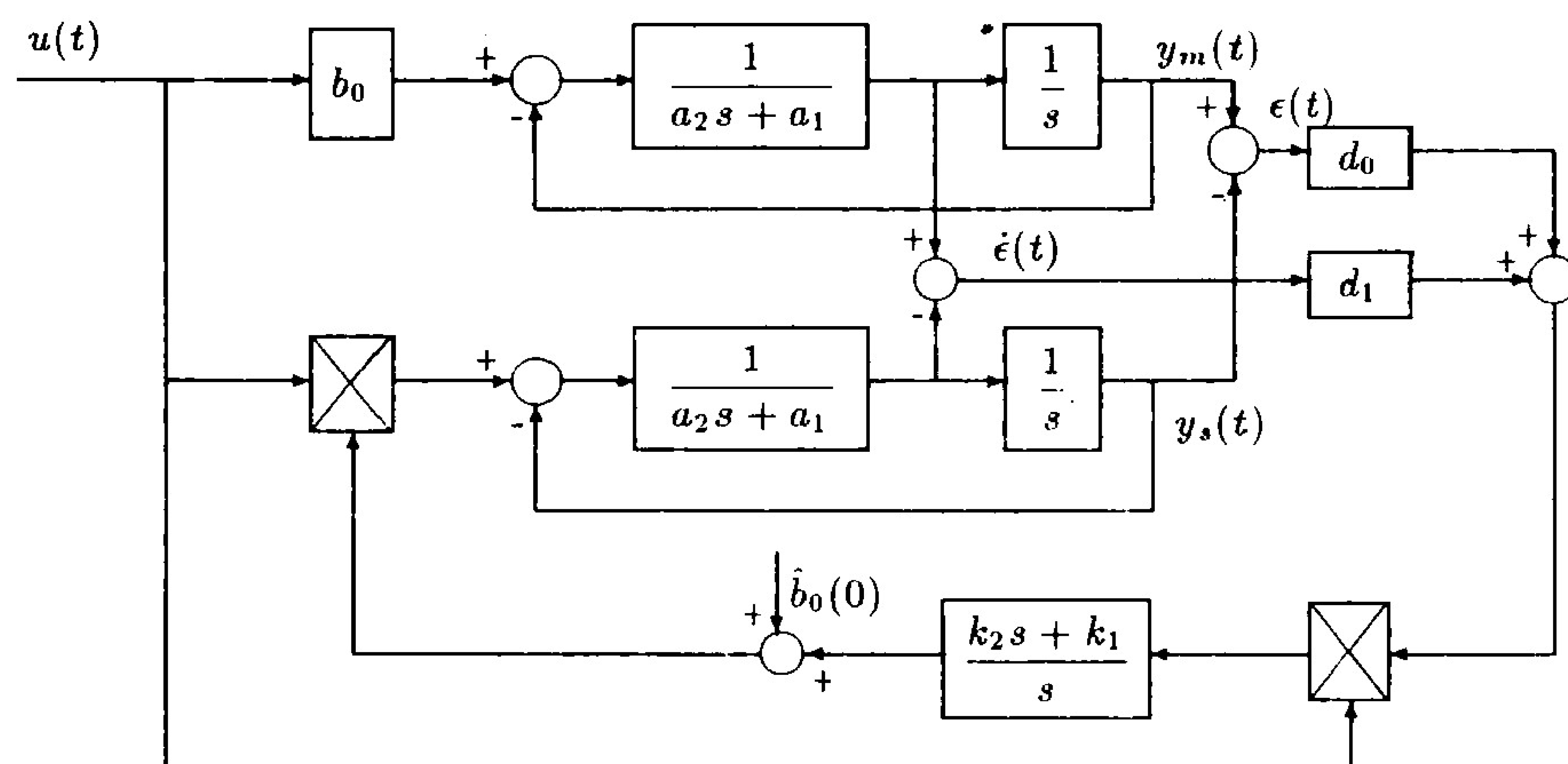


图5-41 模型参考自适应系统的框图

改变自适应控制的增益 $\hat{b}_0(t)$，使得系统的输出可以跟踪参考模型的输出。由框图可见该系统为非线性系统，所以采用 SIMULINK 这类软件对之进行仿真分析是比较合适的。

根据给出的系统框图可以容易地建立起系统的 SIMULINK 模型，如图 5-42 所示，在图中对对象

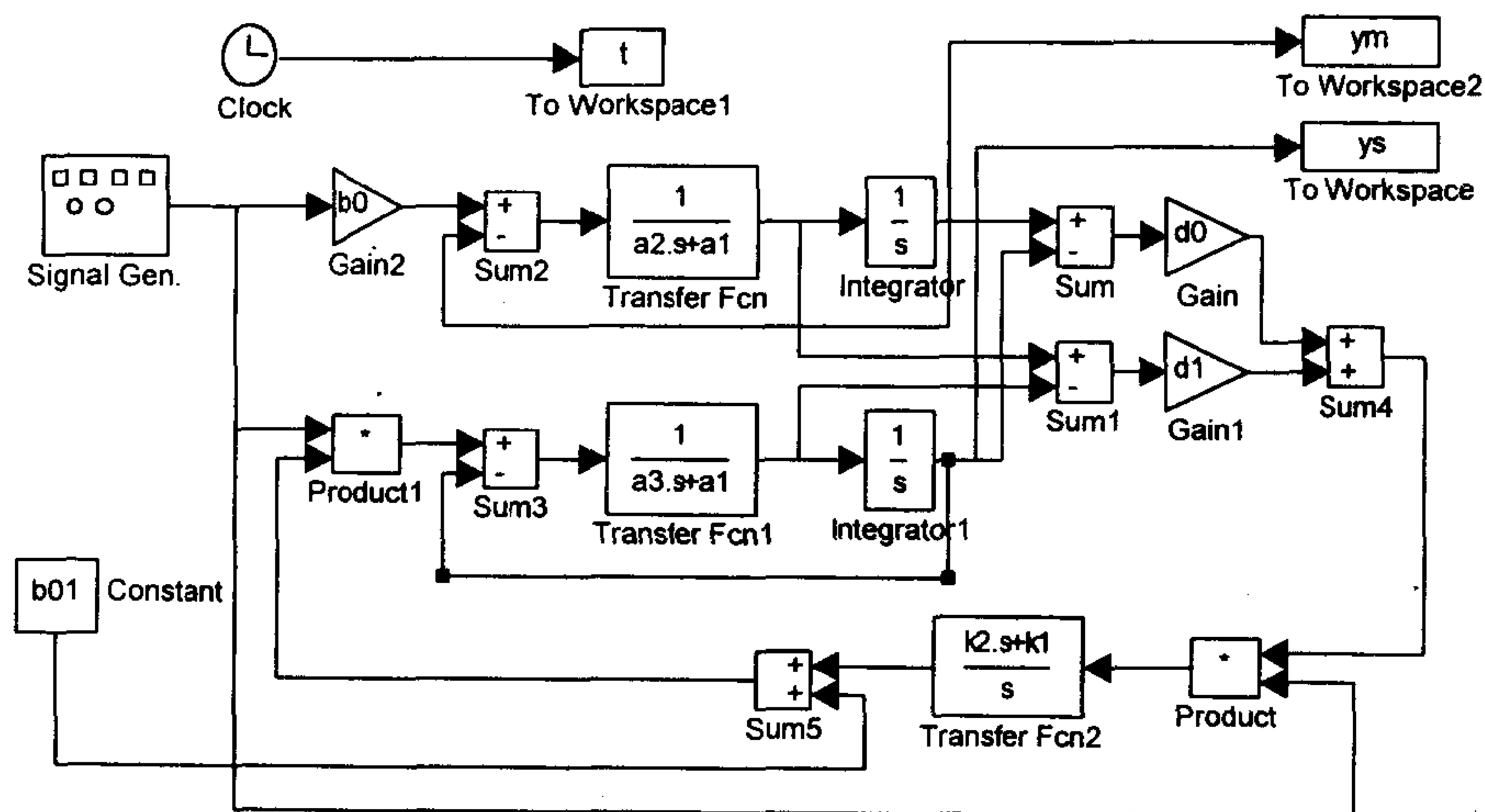


图5-42　模型参考自适应系统的 SIMULINK 表示

模型的一阶环节模型作了修正，将它从原来的 $1/(a_2s+a_1)$ 改写成 $1/(a_3s+a_1)$，若系统参数 $b_0=0.5$, $a_1=0.447$, $a_2=0.1$，且选择控制器参数 $d_0=1$, $d_1=0.5$, $k_1=0.03$, $k_2=1$，若取 $\hat{b}_0(0)=0.2$，输入信号为方波信号且其幅值为 10，频率为 1，并将仿真范围设置为 0~15 秒，这样就可以进一步再调整系统模型的 a_3 参数，使之在 0.02, 0.1 $(=a_2)$, 1, 2, 5, 10 的范围内变化，分别对这些情况进行仿真，则可以得出如图 5-43 所示的仿真结果，其中幅值趋于最小的为参考模型的输出曲线，随着 a_3 值的增

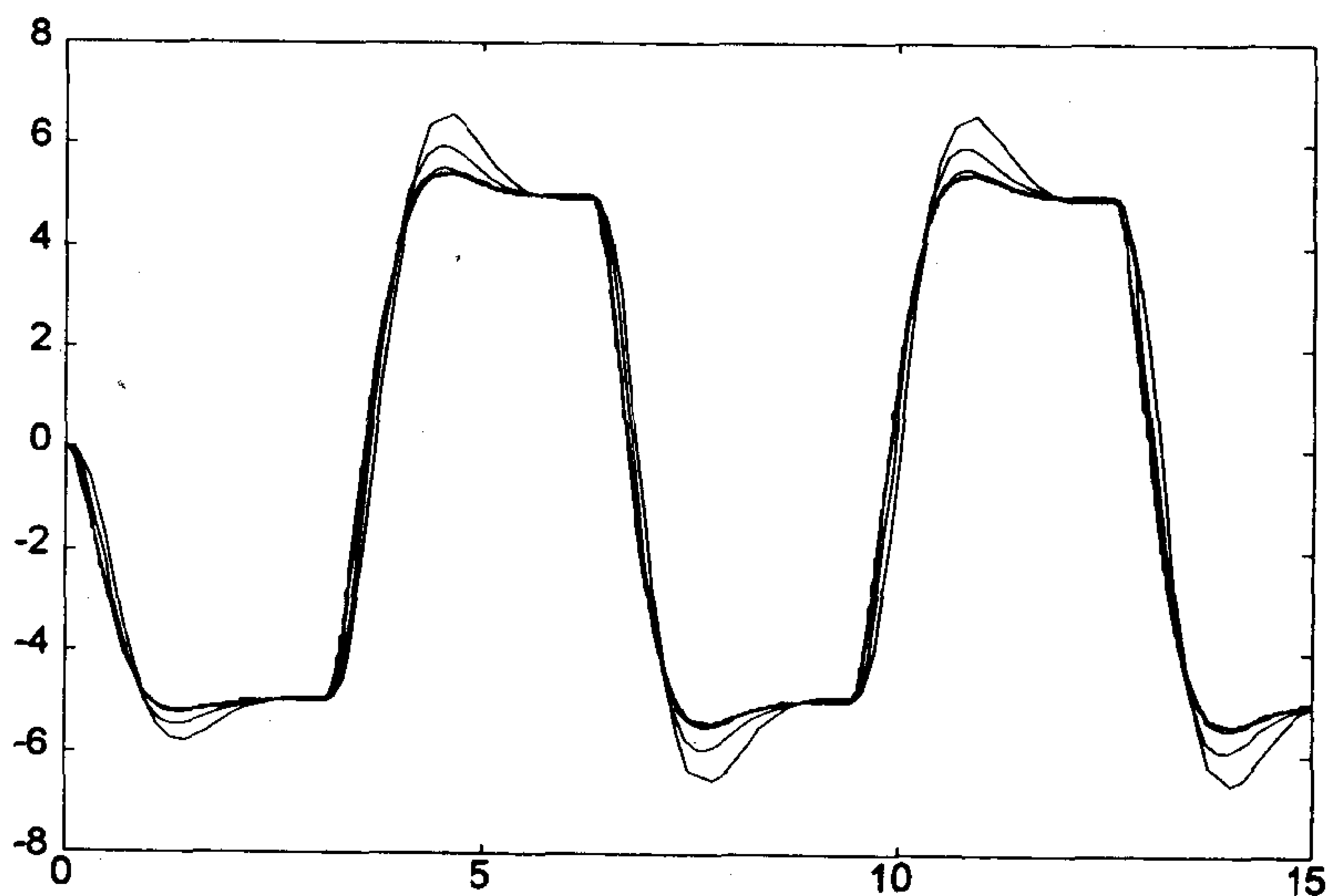


图5-43　模型参考自适应系统的仿真结果

大，自适应控制的效果变坏，但始终保持在可以接受的范围内。由此可见，尽管有时系统的数学模型和参考模型有较大的差异，利用这样的控制策略仍可以获得较满意的结果。

还可以通过仿真的方法对这样的自适应策略进行进一步分析，假设这时系统的模型不再是 $b_0/(a_2s^2+a_1s+1)$，而是具有相对阶 (pole-zero excess, 即分母与分子多项式阶次的差) 为 2 (等于参考模型的相对阶) 的高阶模型，例如 $G(s)=(7s^2+24s+24)/(s4+10s^3+35s^2+50s+24)$，则可以由图

5-44 中的方式用 SIMULINK 构造 $y_s(t)$ 和 $\dot{y}_s(t)$ 信号，再将这个模块镶嵌在图 5-42 中的相应位置，

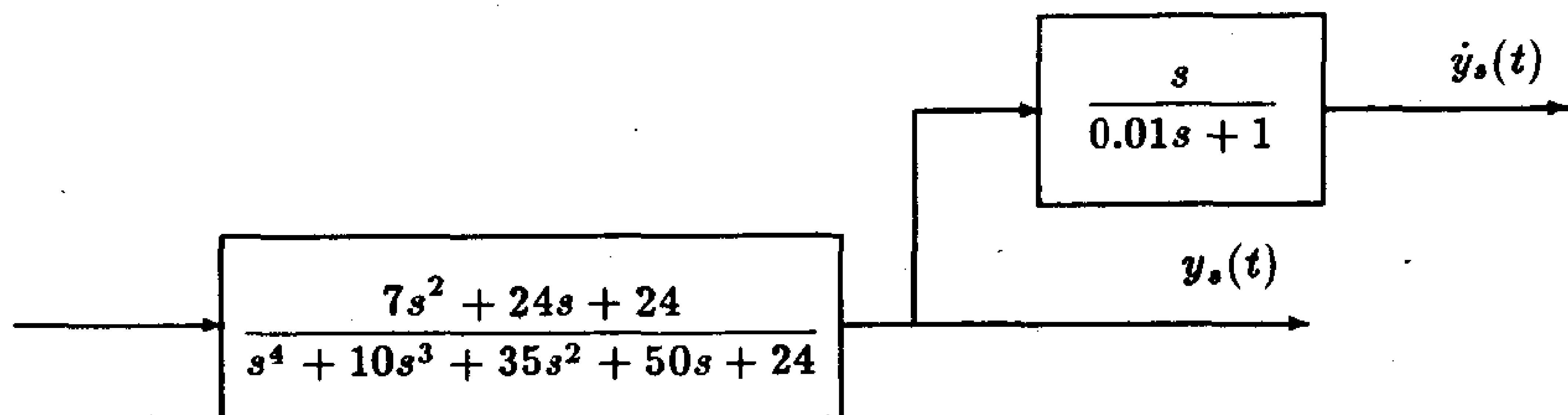


图5-44 系统模型部分的 SIMULINK 表示

则可以得出如图 5-45 所示的仿真结果，可以看出，这样得出的仿真结果仍然是令人满意的。

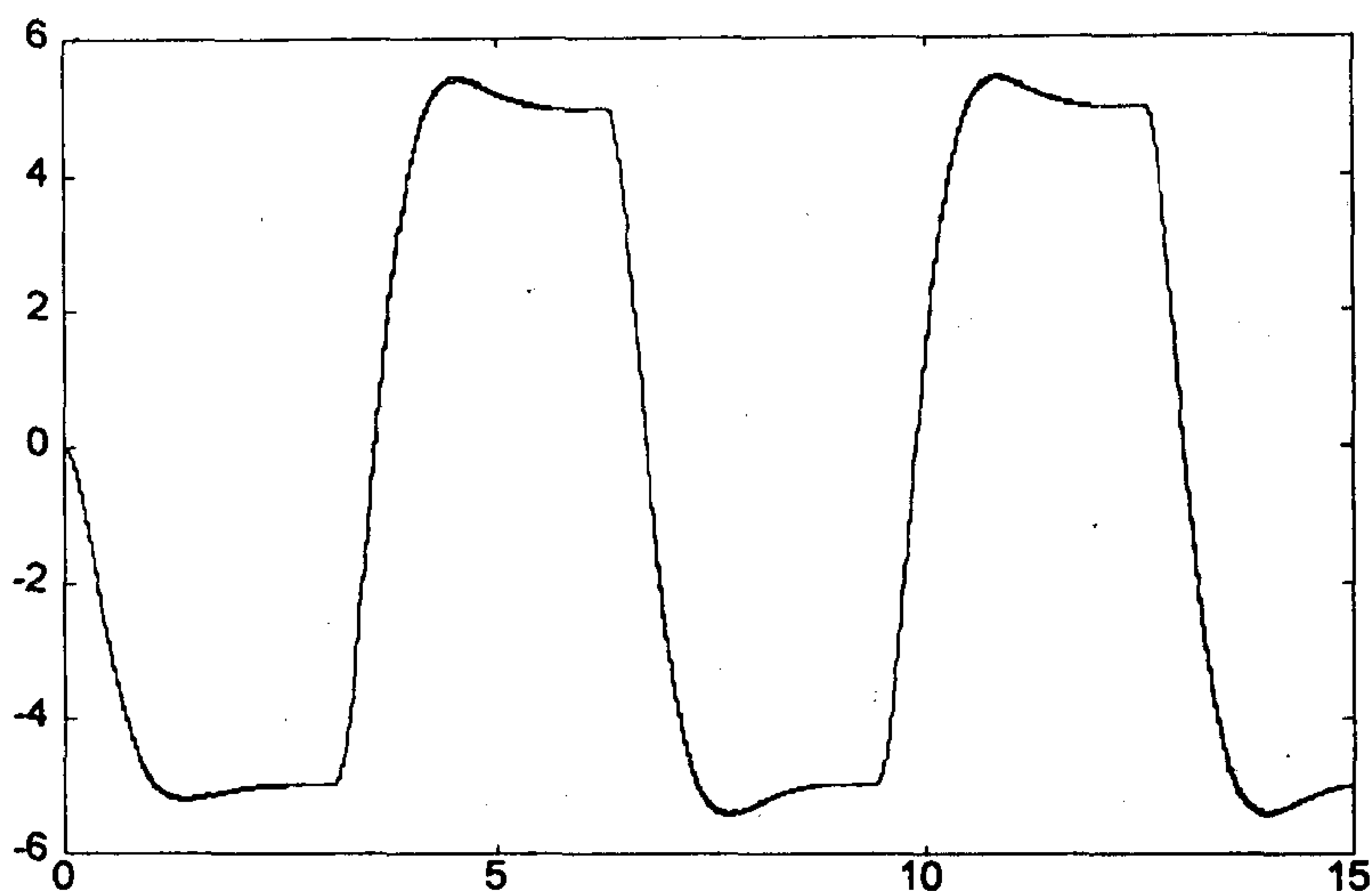


图5-45 系统模型变化时的仿真结果

5.6 连续随机输入系统的仿真算法

5.6.1 传统仿真方法的不适用性

考虑下面给出的一阶线性模型 $G(s)=1/(as+1)$，a 为给定常数。假设输入为 Gauss 白噪声信号 $\gamma(t)$，其均值为 0，方差为 σ^2，可以证明[1]，输出函数 $y(t)$ 亦为 Gauss 信号，其均值为 0，方差为 $\sigma_y^2=\sigma^2/(2a)$。

要仿真此系统，则很自然地可以采用 SIMULINK 环境建立起如图 5-46 所示的系统模型，对此系统进行 30000 个点定步长仿真，在对仿真结果进行分析将得出如下的结论

```
>>  T=0.01; [t,x,y]=euler('randsim',30000*T,[],[1e-3,T,T]);
>>  [mean(y) cov(y)]
ans =    -0.0010      0.0049
>>  T=0.1; [t,x,y]=euler('randsim',30000*T,[],[1e-3,T,T]);
```

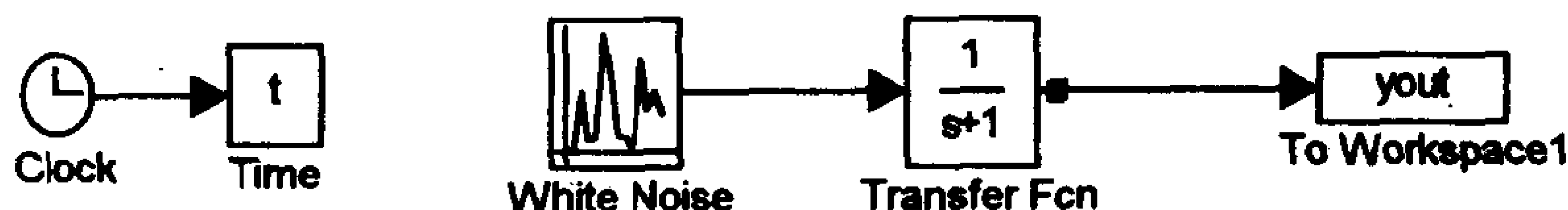


图5-46　随机输入模型的 SIMULINK 表示

```
>>  [mean(y), cov(y)]
ans =    -0.0009     0.0522
```

事实上可以从结果看出，系统输出的方差并不是趋于一个定值。在理论上的分析也可以证明这一点。如果假设在一个计算步长内输入信号为一常数 e_k，并直接对此系统进行离散化，则可以得出系统的模型为

$$y_{k+1}=\mathrm{e}^{-\Delta t/a}y_k+(1-\mathrm{e}^{-\Delta t/a})\sigma e_k$$

式中 Δt 为计算步长， e_k 为满足标准正态分布 $N(0,1)$ 的伪随机数。这时可以得出

$$E[y_{k+1}^2]=\mathrm{e}^{-2\Delta t/a}+2\sigma\mathrm{e}^{-\Delta t/a}E[e_ky_k]+\sigma^2(1-\mathrm{e}^{-\Delta t/})^2E[e_k^2]$$

若输入与输出信号均为平稳过程，则 $E[y_{k+1}^2]=E[y_k^2]=\sigma_y^2$，此外由于 y_k 与 e_k 是相互独立的，则 $E[y_ke_k]=0$ 。这时可以证明

$$\sigma_y^2=\frac{\sigma^2(1-\mathrm{e}^{-\Delta t/a})^2}{(1-\mathrm{e}^{-2\Delta t/a})}=\frac{\sigma^2(1-\mathrm{e}^{-\Delta t/a})}{1+\mathrm{e}^{-\Delta t/a}}$$

若 $\Delta t/a\to0$, 对上式的分子和分母分别作幂级数近似，则可以得出

$$\sigma_y^2=\frac{\Delta t/a+o[(\Delta t/a)^2]}{2+o(\Delta t/a)}\sigma^2\overset{\Delta t\to0}{\rightarrow}\frac{\Delta t}{2a}\sigma^2 \tag{5.6.1}$$

可见，这样得出的输出函数的方差取决于计算步长 Δt，这一结果是不正确的。由此可见，如果输入函数为随机信号时，不能采用传统的方法进行仿真。从这里还可以得出如下的近似结论：当 $\Delta t\to0$ 时，可以对普通随机输入信号进行 $\sqrt{1/\Delta t}$ 加权，这样就会使得输出信号的方差趋近于其理论值。

5.6.2　线性系统的仿真算法

假设线性连续系统的状态方程模型为

$$\dot{x}(t)=Ax(t)+B[d(t)+\gamma(t)],\quad y(t)=Cx(t) \tag{5.6.2}$$

式中 $A\in I\!R^{n\times n}$, $B\in I\!R^{n\times m}$, $C\in I\!R^{r\times n}$, $d(t)\in I\!R^{m\times1}$ 为确定性输入向量， $\gamma(t)\in I\!R^{m\times2}$ 为 Gauss 白噪声向量，满足下式

$$E[\gamma(t)]=0,\quad E[\gamma(t)\gamma^T(t)]=V_\sigma\delta(t-\tau) \tag{5.6.3}$$

定义一个变量 $\gamma_c(t)=B\gamma(t)$, 则可以证明 $\gamma_c(t)$ 亦为 Gauss 白噪声，满足

$$E[\gamma_c(t)]=0,\quad E[\gamma_c(t)\gamma_c^T(t)]=V_c\delta(t-\tau)$$

其中 $V_c=BV\sigma B^T\in I\!R^{m\times m}$ 为一个协方差矩阵，这时式 (5.6.2) 可以改写成

$$\dot{x}(t)=Ax(t)+Bd(t)+\gamma_c(t),\quad y(t)=Cx(t) \tag{5.6.4}$$

状态变量的解析解可以写成

$$x(t)=\mathrm{e}^{-At}x(t_0)+\int_{t_0}^{t}\mathrm{e}^{A(t-\tau)}d(\tau)B\mathrm{d}\tau+\int_{t_0}^{t}\gamma_c(t)\mathrm{d}\tau \tag{5.6.5}$$

假设 $t_0=k\Delta t$, $t=(k+1)\Delta t$, 其中 Δt 为计算步长，并假定在一个计算步长之内确定性输入信号 $d(t)$ 为一个常数，亦即，如 $\Delta t\leq t\leq(k+1)\Delta t$ 时有 $d(t)=d(k\Delta t)$, 则式 (5.6.5) 的离散形式可以写成

$$x[(k+1)\Delta t]=Fx(k\Delta t)+Gd(k\Delta t)+\gamma_d(k\Delta t),\quad y(k\Delta t)=Cx(k\Delta t) \tag{5.6.6}$$

式中 $F=\mathrm{e}^{A\Delta t}$, $G=\int_0^{\Delta t}\mathrm{e}^{A(\Delta t-\tau)}B\mathrm{d}\tau$, 且

$$\gamma_d(k\Delta t)=\int_{k\Delta t}^{(k+1)\Delta t}\mathrm{e}^{A[(k+1)\Delta t-\tau]}\gamma_c(t)\mathrm{d}\tau=\int_0^{\Delta t}\mathrm{e}^{At}\gamma_c[(k+1)\Delta t-\tau]\mathrm{d}\tau \tag{5.6.7}$$

可见矩阵 F,G 和确定性系统的离散化形式是一样的，所以会很容易地求得，但可以看出，若系统含有随机输入时，系统的离散化形式与传统形式是不同的。可以证明 $\gamma_d(t)$ 亦为 Gauss 白噪声向量，且满足

$$E[\gamma_d(k\Delta t)]=0,\quad E[\gamma_d(k\Delta t)\gamma_d^T(j\Delta t)]=V\delta_{kj}$$

式中 $V=\int_0^{\Delta t}\mathrm{e}^{At}V_c\mathrm{e}^{A^Tt}\mathrm{d}t$。

协方差矩阵 V 可以由下述的递推算法求出：

利用 Taylor 级数展开技术可得出

$$V=\int_0^{\Delta t}\sum_{k=0}^{\infty}\frac{R^{(k)}(0)}{k!}t^k\mathrm{d}t=\sum_{k=0}^{\infty}V_k \tag{5.6.8}$$

其中 $R^{(k)}(0)$ 与 V_k 可以由下式递推地求出

$$\begin{cases}R^{(k)}(0)=AR^{(k-1)}(0)+R^{(k-1)}(0)A^T\\ V_k=\dfrac{\Delta t}{k+1}(AV_{k-1}+V_{k-1}A^T)\end{cases} \tag{5.6.9}$$

递推初值为 $R^{(0)}(0)=R(0)=V_c$, $V_0=V_c\Delta t$ 。可以证明 V 为对称正定矩阵，这样可以得出 Cholesky 分解 $V=DD^T$ 。且 $\gamma_d(k\Delta t)=De(k\Delta t)$, 式中 $e(k\Delta t)\in I\!R^{n\times 1}$, 且

$e(k\Delta t)=(e_k,e_{k+1},\cdots,e_{k+n-1})^T$, 使得各个分量 e_k 满足标准正态分布，即 $e_k\sim N(0,1)$ 。系统离散形式的递推解可以写成

$$x[(k+1)\Delta t]=Fx(k\Delta t)+Gd(k\Delta t)+De(k\Delta t),\quad y(k\Delta t)=Cx(k\Delta t) \tag{5.6.10}$$

在仿真时，可以产生一组伪随机数，从而产生向量 $e(k\Delta t)$, 然后求出状态变量 $x[(k+1)\Delta t]$ 并求出输出变量 $y[(k+1)\Delta t]$ 。

可以根据前面给出的算法用 MATLAB 编写出随机线性系统离散化的函数 sc2d(), 其内容如下

```
function [F,G,D] = sc2d(A,B,V,T)
[F,G] = c2d(A,B,T);
V0 = B*V*B'*T; Vd = V0;
vmax = sum(sum(abs(Vd))); vv = vmax;
for i=1:2000
   V1 = T/(i+1)*(A*V0+V0*A'); v0 = sum(sum(abs(V1)));
   Vd = Vd+V1; V0 = V1; vv = [vv v0];
   if v0 < eps, break; end
end
D = chol(Vd);
```

由此函数可以看出，只要给出系统的状态方程 A，B 阵，并给出输入的协方差矩阵 V 和采样周期 T, 则可以得出离散化模型的 F，G，D 矩阵。

重新考虑一下前面给出的一阶系统的例子，系统的状态方程可以写成

$$\dot{y}(t)=-\frac{1}{a}y(t)+\frac{1}{a}\gamma_0(t)$$

系统的输出信号 $y(t)$ 的离散形式可以写成

$$y_{k+1}=\mathrm{e}^{\Delta t/a}y_k+\sigma\sqrt{\frac{1}{2a}\left(1-\mathrm{e}^{-2\Delta t/a}\right)}e_k$$

仿照前面的方法可以证明 $\sigma_y^2\to\sigma^2/(2a)$, 这一方差与理论值是一致的。此外，和前面所用的传统方法不同，用此方法得出的仿真结果的统计量与仿真步长的选取无关。应用此仿真算法也可以对不同的仿真步长得出其均值与方差及其假设检验统计量 [12]，如表 5-4 所示。由表 5-4 可以看出，本仿真方法得出的结果是可信的。

5.6.3 近似仿真方法

由前面的叙述可知，上节中给出的仿真方法只可以用于线性系统的处理，而不能接用于非线性系统，尽管对某一类非线性系统利用前面给出的方法可以推导出仿真似解 [12]，但很难得出一般非线性系统的仿真解法，所以在这一节中将讨论另一种的近似方法。为了得出系统的近似解，必须使用定步长的仿真方法，并对伪随机输入数进行比例化。假定比例化系数为 K_σ ，即 $\gamma_i=K_\sigma e_i$ ，式中 e_i 为满足标准正态分布的

表5-4　仿真结果的统计分析与假设检验

计算步长 Δt	均值 $\bar{y}$	方差 $\hat{\sigma}^2$	均值假设检验 $\eta_{\bar{y}}$	方差假设检验 $\eta_{\hat{\sigma}^2}$	正态性假设检验 η_N
0.02	-2.0753×10^{-6}	1.4305	-2.9350×10^{-4}	-5.6745	7.7593
0.05	-5.5590×10^{-4}	1.4338	0.0786	-5.4015	2.0494
0.10	6.3233×10^{-4}	1.4626	0.0894	-3.0512	0.9768
0.20	4.0403×10^{-4}	1.4913	0.0571	-0.7089	0.5862
0.50	1.4048×10^{-4}	1.5058	0.0199	0.4699	-1.7340

伪随机数，而γ_i 为可以用于仿真的实际输入量。假定在一个计算步长内输入信号 γ_i 为一常值，即

$$\gamma(t)=\gamma_i=K_\sigma e_i,\ \ i\Delta t\leq t<(i+1)\Delta t \tag{5.6.11}$$

为保证伪随机变量 γ_i 的强度与原随机信号 $\gamma_0(t)$ 相同，则可以证明 [12]

$$K_\sigma=\sigma\sqrt{\frac{1}{\Delta t}} \tag{5.6.12}$$

考虑前面的一阶系统的例子，对输入函数进行加权后，系统的仿真模型可以写成

$$y_{k+1}=\mathrm{e}^{-\Delta t/a}y_k+\sigma\sqrt{\frac{1}{\Delta t}}(1-\mathrm{e}^{-\Delta t/a})e_k$$

比较一下前面给出的例子，不难看出当 Δt 很小时，

$$\frac{\sigma\sqrt{\dfrac{1}{\Delta t}}(1-\mathrm{e}^{-\Delta t/a})}{\sigma\sqrt{\dfrac{1}{2a}}(1-\mathrm{e}^{-2\Delta t/a})}=\sqrt{\frac{2a(1-\mathrm{e}^{-\Delta t/a})}{\Delta t(1+\mathrm{e}^{-\Delta t/a})}}=\sqrt{\frac{2a[\Delta t/a+o(\Delta t^2)]}{\Delta t[2+o(\Delta t)]}}\xrightarrow{\Delta t\to 0}1$$

即这一方法在 Δt 很小时得出的统计结果是很接近理论值的。采用此仿真方法对不同的步长所得出的仿真结果的均值和方差以及假设检验的结果如表 5-5 所示。可见，在步长不是很大时所得出的结果是可信的，虽然方差假设检验的结果并不是十分理想。

这一仿真方法的优点是它并不局限于对线性系统的仿真，它完全可以直接用于非线性系统的仿真。这一算法实质上就是用一个能近似保持原噪声信号的强度的比例化的伪随机信号来取代原信号，并将之用于系统的仿真。这时使用者可以采用任何定步长的仿真算法来模拟系统。作为最简单的仿真方法， Euler 法可以方便地用于系统的仿真，这种方法的计算量是很小的，但由于该方法本身的局限性，这种方法并不能保证很高的精度。我们知道， Runge-Kutta 法的精度要高于 Euler 方法，然而，在每个计算步长中要对输入函数进行多次求值，显然在一个计算步长之内，输入信号发生变化，会给这里引入的比例化系数 K_σ 带来麻烦。所以采用此类方法时应格外注意，必须在比例化后的伪随机信号 γ_i 后面跟一个零阶保持器，以保证在计算步长内输入量始终为一常数。

表5-5 仿真结果的统计分析与假设检验

计算步长 Δt	均值 $\bar{y}$	方差 $\hat{\sigma}^2$	均值假设检验 $\eta_{\bar{y}}$	方差假设检验 $\eta_{\hat{\sigma}^2}$	正态性假设检验 η_N
0.02	8.4025×10^{-8}	1.4448	1.1883×10^{-6}	-4.5100	6.7938
0.05	4.0632×10^{-4}	1.4711	0.0575	-2.3605	1.2797
0.10	4.5481×10^{-4}	1.5417	0.0643	3.4008	0.4793
0.20	2.4612×10^{-4}	1.6601	0.0348	13.0691	0.3849
0.50	4.9850×10^{-4}	2.0072	0.0071	41.4094	0.5757

SIMULINK 中也依照这一方法给出了白噪声输入的模块，该模块位于 Extras 组中的 Sources 子模块组中，用户可以直接使用此模块来对随机输入循环进行仿真。

5.7 非线性系统的频率响应分析

非线性系统的频率响应分析和线性系统的是不同的，频率响应分析的结果不但取决于输入信号的频率，还取决于输入信号的幅值 [2]，对线性系统来说，因为它满足叠加原理，所以频率响应只取决于输入信号的频率，而和其幅值没有关系，因此对线性系统来说，用 Bode 图就可以较好地表述出系统的频率响应。

在非线性系统的设计中有时确实需要其频率响应数据 [8]，在实际设计中，有很多方法可以求取系统的频率响应，例如对含有可以解析描述的非线性系统来说，可以采用 Volterra 序列来求取频率响应 [9]，而对实际系统来说可以由仪器来测出系统的频率响应。在本节中将介绍一种通过仿真来求取频率响应的方法，此方法不但能够获得含有解析非线性环节系统的频率响应，还可以求出带有分段线性的非线性环节系统的频率响应，并将给出求取频率响应的 MATLAB 实用程序。

5.7.1 仿真分析算法

用于求取频率响应的仿真模型如图 5-47 所示，其中输入函数为 $r(t) = A_a \sin(\omega t)$，

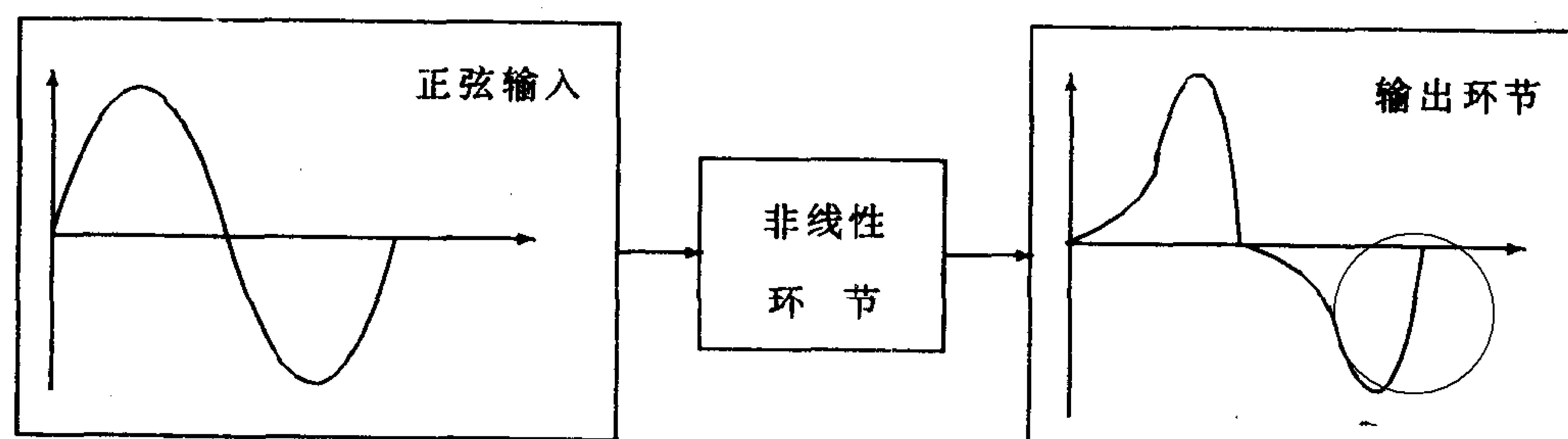


图5-47 频率响应仿真框图表示

其中变量 A_a 和 ω 可以在 MATLAB 环境中用变量 aa 和 ww 来表示。在仿真过程中可以采用定步长 4/5 阶 Range-Kutta 方法，其中输入信号 $r(t)$ 为正弦信号，满足

$$r(t) = A_i \sin(\omega_j t),\quad i = 1, 2, \ldots, n_a,\quad j = 1, 2, \ldots, n_\omega \tag{5.7.1}$$

如果假定在每个正弦周期 $T = 2\pi/\omega$ 内仿真 n 个点，则可以由下式确定仿真步长

$$\Delta t_j = \frac{T}{n-1} = \frac{2\pi}{(n-1)\omega_j} \tag{5.7.2}$$

在下面的研究中，取 $n = 50$，可以看出，若研究的非线性系统为稳定的，则在有限次的仿真之后，例如经过 p 个周期之后，系统响应的下两个周期的曲线趋于一致。若输入的信号为 (A_i, ω_j) 对，则可以由下式计算出 $\hat{A}_{i,j}$

$$\hat{A}_{i,j} = \frac{\max\limits_q(y_{p,q}) - \min\limits_q(y_{p,q})}{2A_i} \tag{5.7.3}$$

可见，在每个周期内将至少有两个过零点，如果有多于两个过零点，则存在着求取相应相位值的问题，如果只有两个过零点，就要取出处于上升段处的相位值。假设这样的过零点发生在第 k 和 $k+1$ 仿真点处，则可以通过线性插值得出相位值 $\phi_{i,j}$

$$\phi_{i,j} = \frac{2\pi\left(k - \dfrac{y_{p+1,k}}{y_{p+1,k+1}}\right)\Delta t_j}{2\pi/\omega_j} = \left(k - \frac{y_{p+1,k}}{y_{p,k+1}}\right)\frac{360^\circ}{(n-1)} \tag{5.7.4}$$

这一算法看起来似乎很简单，但是即使对一个极简单的系统来说这样的频率响应也需要几百个周期的仿真分析，所以对要很多频率值和幅值来说，计算量还是相当大的，故为了提高计算效率，可以采用下面的措施：

- **避免不必要的计算周期**：定义一个收敛准则，使得当满足收敛准则时，则终止对此 (A_i, ω_j) 的仿真，存储其得出的幅值与相位值，再开始对下一个 (A_i, ω_j) 对的仿真。收敛准则可以定义为

$$J = \frac{\max\limits_q |y_{p,q} - y_{p+1,q}|}{\max\limits_q |y_{p+1,q}|} \leq \varepsilon \tag{5.7.5}$$

  式中 $q = 1, 2, \ldots, n-1$ 为第 p 周期内的所有仿真点，ε 为指定的误差限，例如取 $\varepsilon = 0.02$。

- **使用在 (A_i, ω_{j-1}) 对时收敛处的状态值作仿真初值**：在实际仿真中，系统的收敛速度取决于初始状态的选取，所以 ω_{j-1} 时的收敛状态可能对求取下一个频率下的响应有用，因为在频率的变化值不是很大时，(A_i, ω_{j-1}) 的值也不会有太大的变化，故可以用它来作为下一个仿真过程的初值。

5.7.2 初始状态向量的较精确近似

在每个仿真周期内都要从一个初始状态向量出发去对该系统进行仿真，直至得出收敛的解，而初始状态向量的选择对影响仿真的收敛速度，尤其是当频率 ω_j 增加时，会需要更多的仿真步长才会收敛于一个稳定的输出波形，所以在仿真分析时如果每个周期内的响应都从一个较好的初始状态出发，这无疑会大大地提高收敛的速度，更快地获得非线性系统的频率响应数据。

要获得一个较好的初始状态向量，则首先应得出 SIMULINK 描述的非线性系统的线性化模型，

$$\dot{x} = Ax + bu, \quad y = cx \tag{5.7.6}$$

若 ω_j, $j = 1,2,\cdots,k$ 已经仿真出来，且对该频率系统的幅值 $\hat{A}_{i,j}$ 和相位 $\phi_{i,,j}$ 都已经得出，这样对下一个频率 ω_{k+1} 点处的幅值 $\hat{A}_{i,k+1}$ 和相位 $\phi_{i,k+1}$ 可以由线性预报的方法得出

$$\widetilde{A}_{i,k+1} = \frac{\hat{A}_{i,k}^2}{\hat{A}_{i,k-1}}, \quad 且 \quad \hat{\psi}_{i,k+1} = 2\phi_{i,k} - \phi_{i,k-1} \tag{5.7.7}$$

这时输出 $y(t)$ 可以被假设成正弦信号

$$\hat{y}(t) = \widetilde{A}_{i,k+1}\sin(\omega_{k+1}t + \hat{\phi}_{i,k+1}) \tag{5.7.8}$$

这样该波形的第 γ 阶导数，$\gamma = 0,1,\ldots,n-1$ 可以写成

$$\hat{y}^{(\gamma)}(t) = \widetilde{A}_{i,k+1}\omega_{k+1}^{\gamma}\sin\left(\frac{\gamma}{2}\pi + \omega_{k+1}t + \hat{\phi}_{i,k+1}\right) \tag{5.7.9}$$

$$\hat{y}^{(\gamma)}(0) = \widetilde{A}_{i,k+1}\omega_{k+1}^{\gamma}\sin\left(\frac{\gamma}{2}\pi + \hat{\phi}_{i,k+1}\right) \tag{5.7.10}$$

而输入波形的第 γ 阶导数可以写成

$$r^{(\gamma)}(t) = A_i\omega_{k+1}^{\gamma}\sin\left(\frac{\gamma}{2}\pi + \omega_{k+1}t\right), 且 \; r^{(\gamma)}(0) = A_i\omega_{k+1}^{\gamma}\sin\left(\frac{\gamma}{2}\pi\right) \tag{5.7.11}$$

可以证明，若线性化模型为可观测的，则系统的初始状态向量可以由下式近似得出

$$x(0) = \begin{bmatrix} c \\ cA \\ \cdots \\ ca^{n-1} \end{bmatrix}^{-1} \begin{bmatrix} \hat{y}(0) \\ \dot{\hat{y}}(0) - cbu(0) \\ \cdots \\ \hat{y}^{(n-1)}(0) - \sum_{p=1}^{n-1} cA^{p-1}bu^{(n-p-1)}(0) \end{bmatrix} \tag{5.7.12}$$

5.7.3 频率分析的 MATLAB 程序实现

可以用 MATLAB 语言编写一个程序 nlfreq.m，它可以用来对 SIMULINK 描述的非线性系统模型求取频率响应数据，在运行此程序时用户需要提供模型的文件名、频率和幅值范围等参数，还需要给出每个正弦周期内仿真点的数目，这一主程序的清单如下

```
global aa ww
pnt=input('Enter the number of point in each cycle => ');
modname=input('Enter the model name => ','s');
w=input('Enter the frequency range => ');
amp=input('Enter the amplitude range => ');
maxc=200;
[mag,pha,fmat]=get_freq(modname,w,amp,pnt);
```

在此程序中调用了 MATLAB 函数 get_freq.m，此函数可以取出给定系统的频率响应数据，通过此函数可以用仿真的方法进行频率响应分析，该函数的调用格式为

[mag,pha,fmat]=get_freq(modname,w,amp,pnt,maxc)

其中 mag，pha 为指定频率 w 范围内的幅值和相位构成的矩阵，它们是根据不同输入幅值 amp 而进行存储的，modname 为 SIMULINK 模型的文件名，且在该模型中输入信号定义为输入接口 (即 import 模块) 与正弦信号发生器之和，pnt 为每个周期内指定的仿真点数。

```
function [mag,pha,fmat]=get_freq(modname,w,amp,pnt)
global ww aa
fmat=[]; ic=[]; maxc=200; dw=2*pi/(pnt-1); t0=0; dt=0.1; tol=0.0001;
[a,b,c,d]=linmod(modname); x0=zeros(length(c),1); xz=x0;
for jm=1:length(amp);
   aa=amp(jm); disp(['Amplitude=', num2str(amp(jm)),', please wait']);
   for im=1:length(w)
      i0=0; e0=1e100; icc=0; ww=w(im); dt=dw/ww; T=2*pi/ww;
      t=0:dt:T; y0=aa*sin(ww*t)'; icount=0; an=aa; w0=ww;
      if (im>2),
         a0=mag(im-2:im-1); p0=pha(im-2:im-1); x0=get_inix(a,b,c,an,a0,w0,p0);
      end
      while i0==0
         [t,x,y]=rk45(modname,[t0 T],x0,[tol dt dt]); err=y-y0;
         dd=max(abs(err)); icount=icount+1; d0=dd/max(abs(y)); x0=x(pnt,:);
         if (dd>e0), icc=icc+1; end, e0=min([dd,e0]);
         if icc>20, i0=1; a0=aa; w0=ww;
            disp(['System not stable for a=' num2str(a0) ', w=' num2str(w0)])
         end
         if (icount>maxc|d0<0.02), i0=1; else, y0=y; end
      end
      mag(jm,im)=0.5*(max(y0)-min(y0))/aa; fmat=[fmat [y0;y]];
      yx=[y0(2:length(y0)); y0(length(y0))]; ii = find(yx.*y0<0);
```

```
        ic(jm,im)=icount;
        for j0=1:2
           if y0(ii(j0))<0,
              k=abs(y0(ii(j0)))/y0(ii(j0)+1); t1=t(ii(j0)); t2=t(ii(j0)+1);
              pha(jm,im)=-360*(t1+k*(t2-t1)/(1+k))/T;
        end, end
    end, end
```

下面给出求取线性化模型 (a,b,c) 的初始状态向量的函数 get_inix.m，假设正弦输入信号的幅值为 an, 且前两个频率点 w0 处的幅值与相位的值由 a0, p0 给出，这时近似的状态向量初值可以由下面函数得出

```
x0=get_inix(a,b,c,an,a0,w0,p0)
```

这一函数可以首先预报幅值 ax 和相位值 px，并从中得出初始状态向量的值 x0，此函数的清单如下：

```
function x0=get_inix(a,b,c,an,a0,w0,p0)
C=obsv(a,c); Y=[]; u=[]; n=length(c);
ax=a0(2)^2/a0(1); px=pi/180*(2*p0(2)-p0(1));
for i=1:n, u(i)=w0^(i-1)*sin((i-1)/2*pi); end
for i=1:n
   ym(i)=ax*w0^(i-1)*sin((i-1)*pi/2+px); v=0;
   for j=1:i-1, v(j)=c*a^(j-1)*b*u(i-j); end
   Y=[Y;ym(i)-sum(v)];
end
x0=an*inv(C)*Y;
```

例 5.11 考虑典型非线性反馈系统，其中前向通路的线性传递函数为 $1/(s^2+s+1)$, 而非线性环节在负反馈回路，其元件为 $\mathcal{N}(x)=x/2+x^3/6$ 给出的多项式. $A_1=0.1$ 时每个相位点所需要的仿真周期为 $p+1=3,2,3,3,3,3,3,3,3,3,3,4,4,5,6,7,9,10,13,16$ 。该系统的线性化模型可以写成

$$\dot{x}=\begin{bmatrix}-1 & -1.5\\ 1 & 0\end{bmatrix}x+\begin{bmatrix}1\\ 0\end{bmatrix}u,\ \ y=[0\ \ 1]x$$

采用改进的计算方法，则在 $A_1=0.1$ 时每个频率点所需要的仿真周期为 $(p+1)_j=3,2,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,4,4,2,7$ 。可以看出，即使预报的初始状态向量并不是很准确，在高频处还是可以显著地减少仿真周期数目，从而大大提高非线性系统频率响应分析程序的效率，由前面给出的仿真程序可以得出系统的频率响应曲线，如图 5-48 所示。

习　　题

1) 绘制出下面各个系统的频率响应曲线，包括 Bode 图， Nyquist 图及 Nichols 图，并求出各个模型的幅值裕度和相位裕度

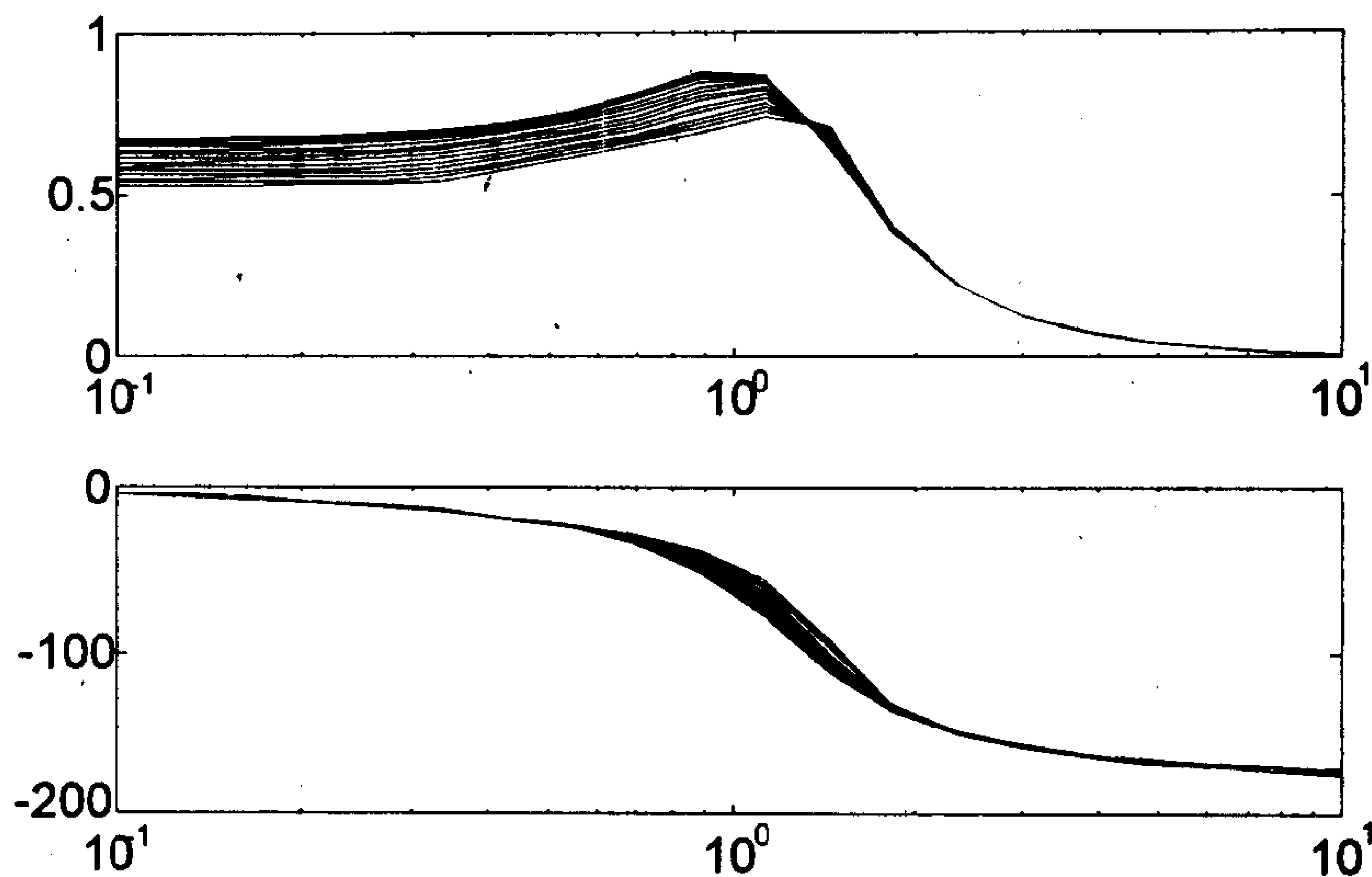


图5-48 非线性系统的频率响应分析

(a) $G(s)=\dfrac{6s^3+26s^2+6s+20}{s^4+3s^3+4s^2+2s+2}$，频率范围 $\omega\in[0.1,10]$

(b) $\dot{x}(t)=\begin{bmatrix}-5 & 2 & 0 & 0\\ 0 & -4 & 0 & 0\\ -3 & 2 & -4 & -1\\ -3 & 2 & 0 & -4\end{bmatrix}x(t)+\begin{bmatrix}1\\2\\0\\1\end{bmatrix}u(t),\quad y(t)=[1,2,3,4]x(t),\ \omega\in[0.01,100]$

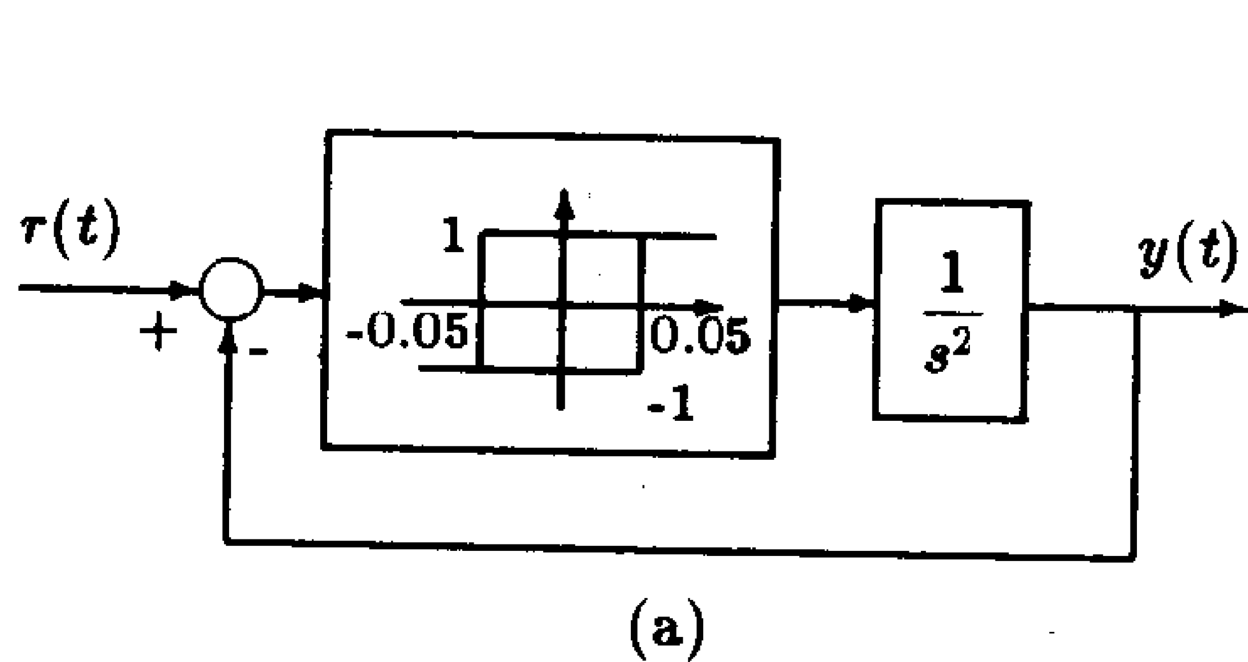
(c) $H(z)=\dfrac{2+3z^{-1}+4z^{-2}}{1+3z^{-1}+3z^{-2}+z^{-3}}$，自动选择频率范围。

(d) $G(s)=\dfrac{1}{(s+1)^3}\mathrm{e}^{-2s}$，$\omega\in[0.1,100]$

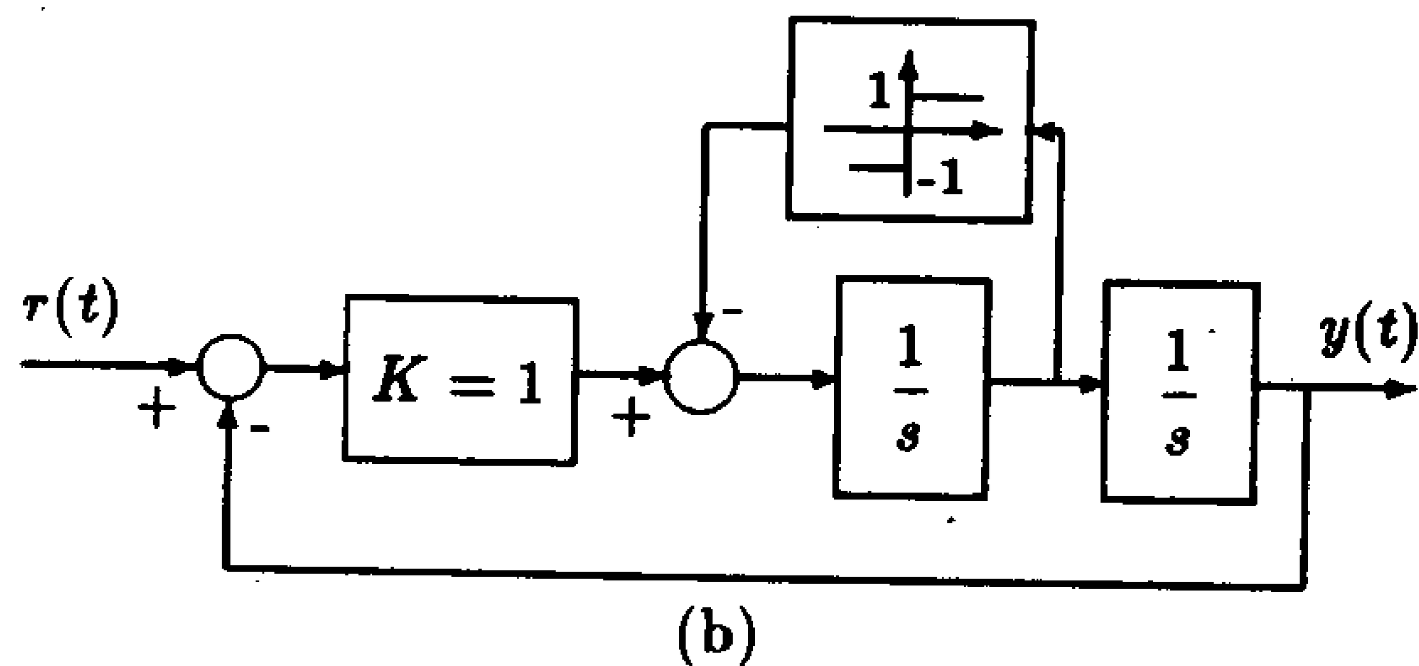
(e) $G(s)=\dfrac{(s+1)^2}{s^3(s^2+1.05s+12.25)}$，选择不同的频率范围并比较结果。

2) 绘制上面各个模型的阶跃响应和脉冲响应曲线。

3) 建立起下面各个框图给定的控制系统 SIMULINK 模型，并在适当的时间范围内对它们进行仿真分析，绘制出在不同阶跃输入幅值下的输出曲线。试分析如果不采用时钟环节，则用 plot() 函数绘制系统响应时会出现什么问题，并对所遇到的问题给出必要的解释。



(a)



(b)

4) 第 6 章将引入传统 PID 控制的改进形式，其数学表示为

$$u(t)=K_p\left[(\beta u_c-y)+\frac{1}{T_i}\int e\mathrm{d}t-T_d\frac{\mathrm{d}y}{\mathrm{d}t}\right]$$

其中 u_c 为参考输入信号，y 为输出信号，而误差信号 $e=u_c-y$。试根据这一公式建立起改进 PID 控制的 SIMULINK 模块，该模块的输入有两个 u_c 和 y，而将 $u(t)$ 作为该模块的输出，且使得用户可以用对话框的方式输入 4 个参数 K_p, β, T_i 和 T_d。

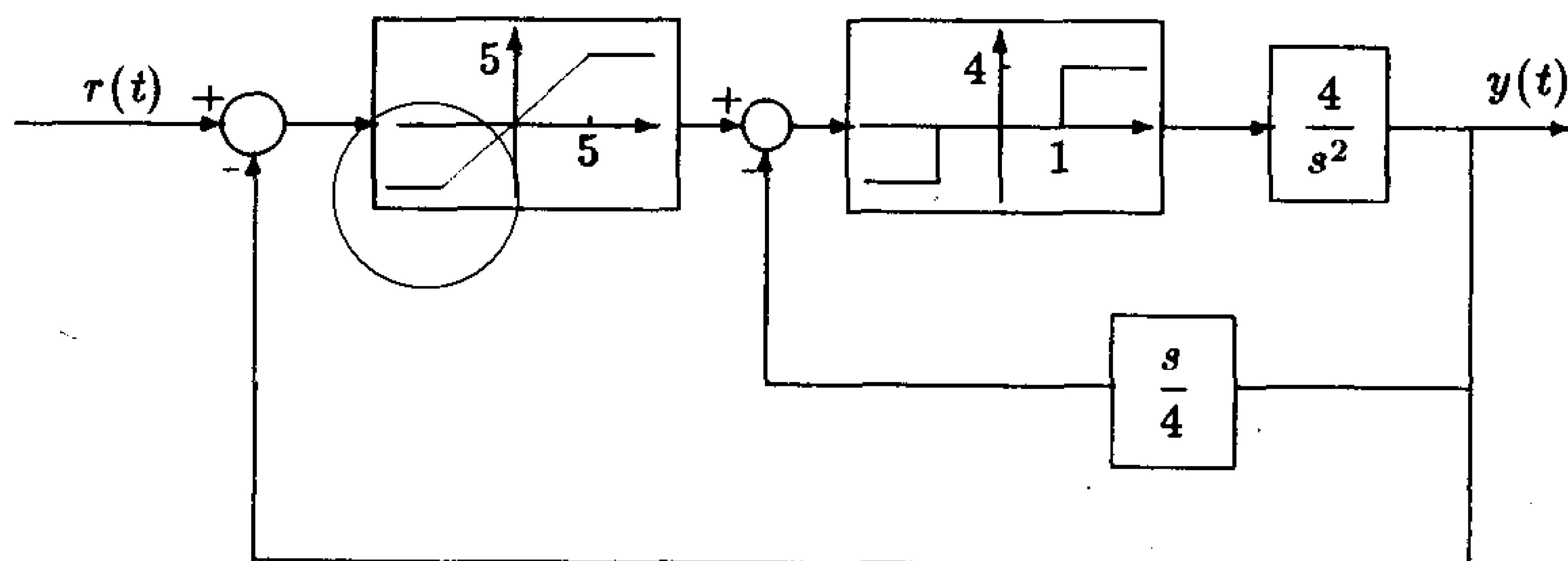
5) 用 SIMULINK 构造带有状态向量输出的状态方程模型封装模块。

6) 用 SIMULINK 建立下面的时变系统模型，并对值进行仿真分析

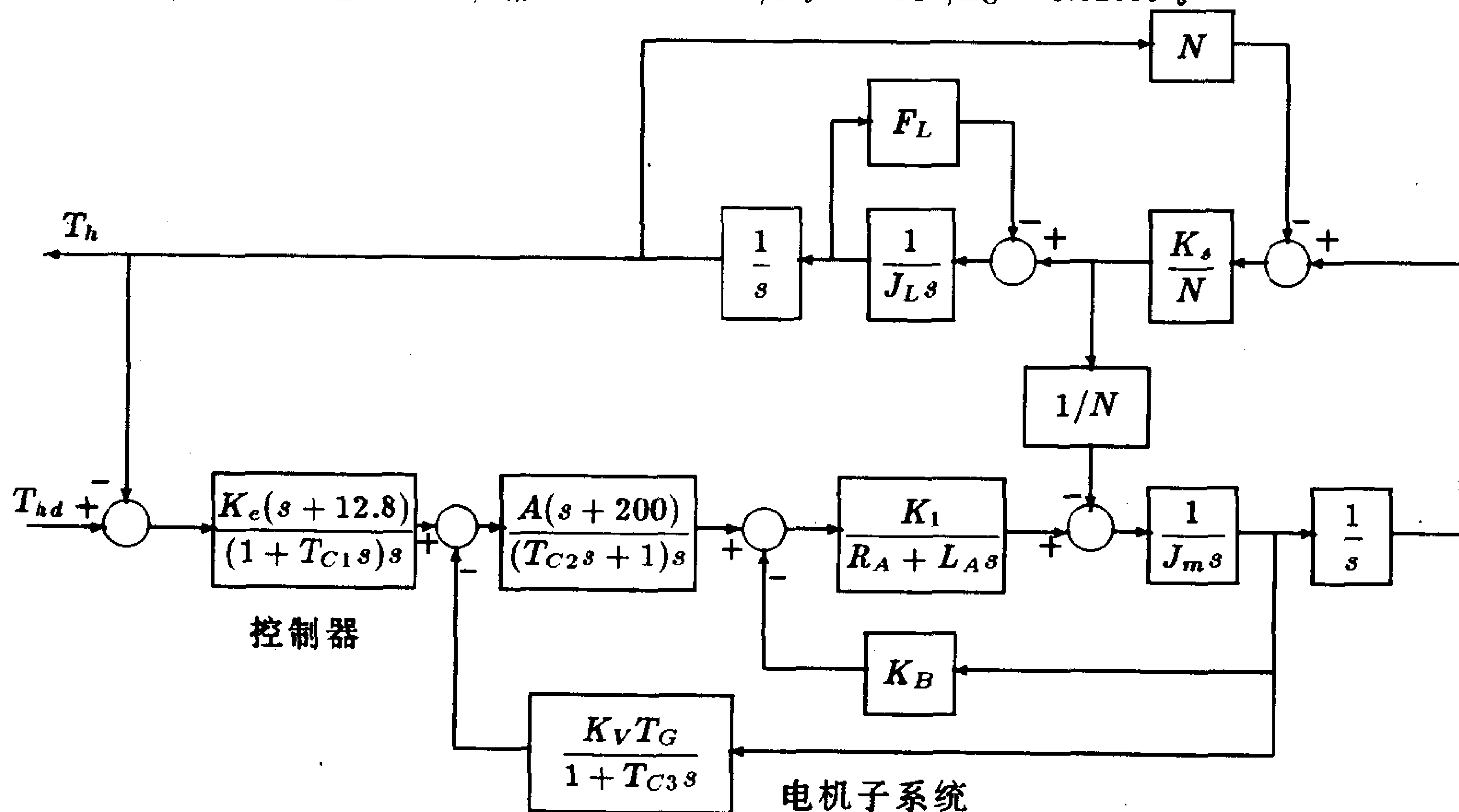
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & t \\ 0 & e^{-\alpha t} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

7) 写出 Lorenz 方程的 S 函数表示，并通过 SIMULINK 对之进行仿真分析并绘制图形。

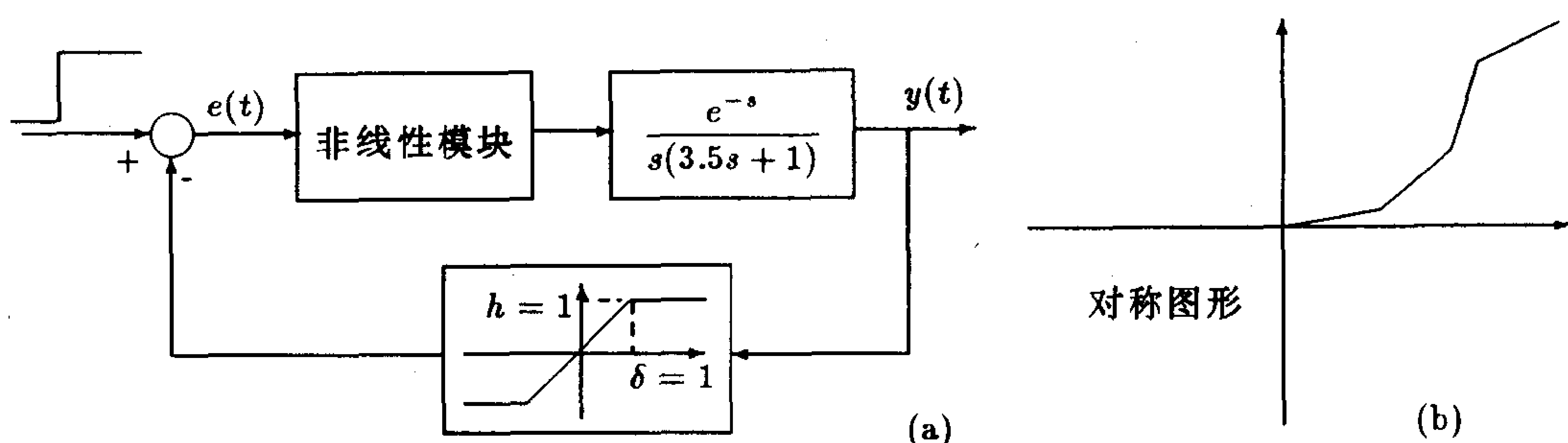
8) 对下图系统进行仿真分析，并令 $x_1=y, x_2=\dot{y}$，作出 x_1-x_2 相平面曲线，观察滑模 (sliding mode) 现象。(提示：可以依照文献 [3] 选择 $r(t)=0$，并取初值 $x_1(0)=-10, x_2(0)=0$。)



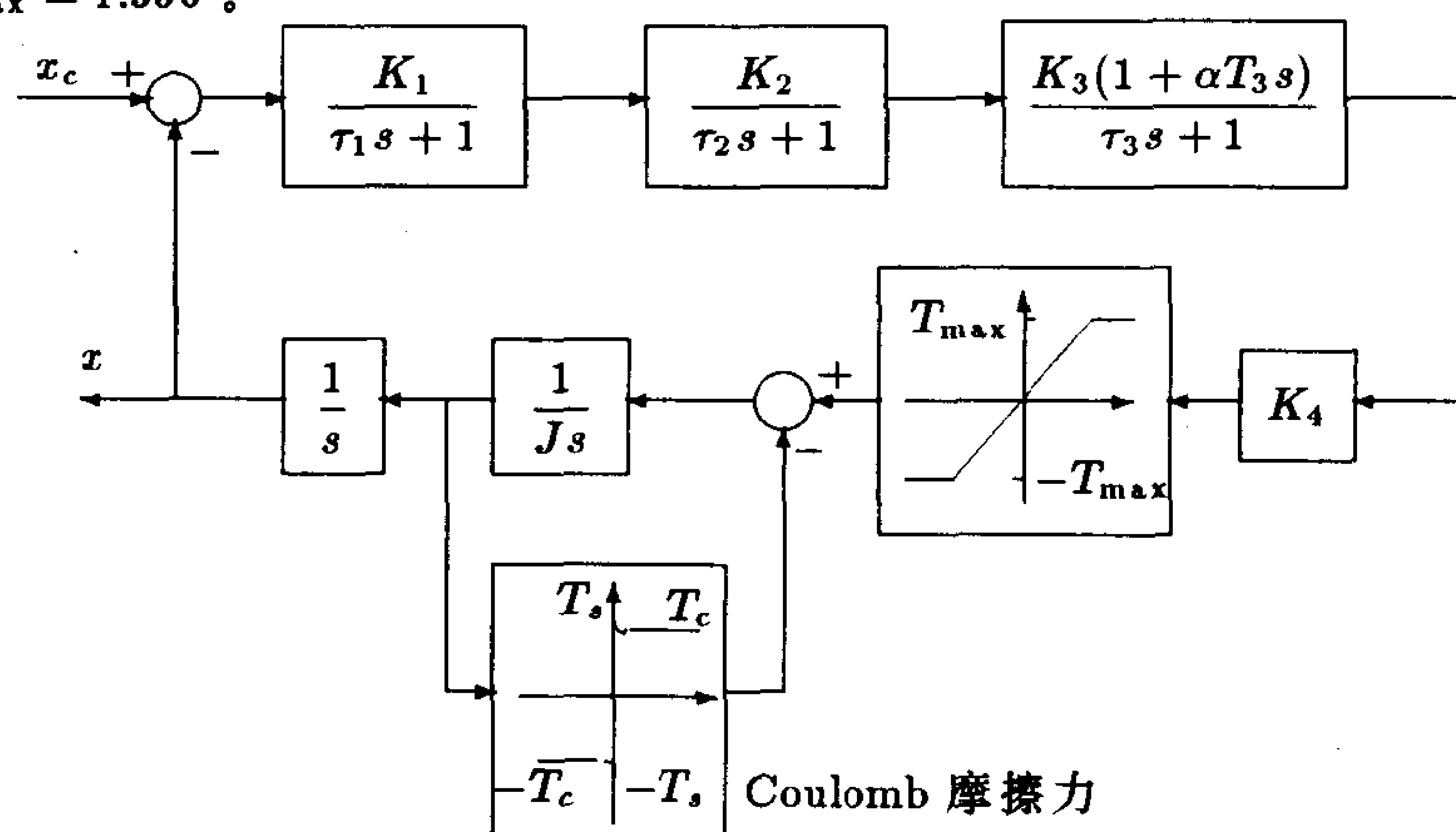
9) 对下图给出的伺服跟踪控制系统 [6] 进行仿真分析，其中系统的常数为 $J_L=0.5, F_l=28.3, K_s=4440.0, N=2500, K_e=27, T_{C1}=T_{C2}=T_{C3}=0.001, A=56, K_1=0.0275, R_a=9, L_A=0.004065, K_B=0.04, J_m=1.71\times10^{-6}, K_v=0.049, T_G=0.02865$。



10) 对下图 (a) 中给出的非线性系统进行仿真分析并绘制响应曲线，其中非线性系统的曲线如图 (b) 所示。(提示：可以通过查表的方式建立其非线性系统模型)



11) 对下面给出的卫星控制系统模型 [6] 进行仿真研究，其中系统的常数为 $K_1=15, \tau_1=0.006, K_2=1000, \tau_2=0.01, K_3=0.1, \tau_3=0.0555, K_4=1.356, \alpha=10, J=271.2, T_s=0.2712, T_c=0.1356, T_{\max}=1.356$ 。



12) 参考第 4 章给出的 F-14 基准测试问题模型，对白噪声输入 $n(t)$ 进行数值仿真，并得出 N_{Z_p} 的方差与均值，并根据仿真数据绘制出该信号的概率密度曲线。(提示：可以根据该信号数据进行变换，由直方图绘制函数得出概率密度)

参 考 文 献

[1] Åström K J. Introduction to stochastic control theory. New York: Academic Press, 1970

[2] Atherton D P. Nonlinear control engineering – describing function analysis and design. London: Van Nostrand Reinhold. 1975

[3] Atherton D P. Stability of nonlinear systems. Herts, England: Research Studies Press (John Wiley & Sons), 1981

[4] Forsythe G E, Malcolm M A, and Moler C B. Computer methods for mathematical computations. Englewood Cliffs: Prentice-Hall, 1977

[5] Goodwin A J, Jobling C P. BlockEdit, block diagram input of model descriptions for Ecstasy, version 2.0β. Control and computer aided engineering research group. University College of Swansea and SERC, 1990

[6] Hay J L, Pearce J G, Turnbull L *et al.* ESL application manual. Salford: ISIM Simulation, 1988

[7] Laub A J. Efficient multivariable frequency response computations. IEEE Trans. Automatic Control, 1981, AC-26(4): 407-408

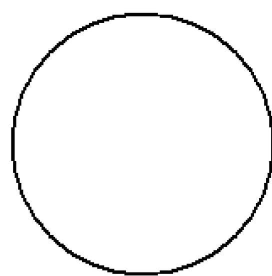
[8] Nanka-Bruce O, Atherton, D P. Design of nonlinear controllers for nonlinear plants. Proceedings 11th IFAC Congress, Vol. 5. Tallinn, USSR, 1989

[9] Peyton-Jones J C, Billings S A. Sheffield nonlinear integrated package – a user/programmer's manual. Sheffield University, 1990

[10] 徐心和. 模型参考自适应系统. 沈阳：东北工学院讲义， 1982

[11] Xue D, Atherton D P. BLOCKM – a block diagram modelling interface and its applications. Proceedings IEEE Symposium on CACSD. Napa. USA, 1992. 242-249

[12] Xue D, Atherton D P. Simulation analysis of continuous systems driven by Gaussian white noise. In: Jamshidi M, Herget C J, eds. Recent advances in computer-aided control systems engineering. Amsterdam: Elsevier Science Publisers B V, 1992. 431-452



第 6 章 控制系统计算机辅助设计 (频域方法)

6.1 引 言

随着计算机技术的飞速发展，控制系统计算机辅助设计技术不但从工具上，而且从理论和算法上也得到巨大的进步，以前被认为很难设计控制器的系统可以由新的方法和控制策略较容易地获得结果。早期控制器的设计往往依赖于试凑的方法，随着计算机技术和软件工具的普及，控制系统计算机辅助设计算法目前越来越适于计算机实现。在很多场合下，用户只需通知计算机已知条件和设计的目标，就可以立即获得所需的控制器和仿真结果。

英国学派基于传递函数矩阵来研究多变量系统的计算机辅助设计问题，后来出现了多变量系统频域设计工具箱等实用工具，其中以 Rosenbrock 教授为代表的 Nyquist 类设计方法及以 MacFarlane 教授为代表的特征轨迹设计方法在多变量系统的频域设计研究中占有主导地位，同样基于频域方法来设计控制系统的还有以色列学者 Horowitz 教授及其合作者们，他们系统地提出了定量反馈理论 (QFT) 来对各种控制系统设计控制器，1993 年美国学者研制出了基于 MATLAB 的 QFT 设计工具箱，从一定程度上解决了该方法的应用问题。与此同时，美国和其它国家的研究者们在处理多变量系统时似乎更看重状态空间方法及使用。

瑞典学者 Karl Åström 教授提出了非常实用的自整定 PID 控制策略，该方法近来也有了较大的进展，并在工业控制上得到了较多的应用。

最优控制理论也有了较多的发展，在多变量系统的设计方法中英国学者 John Edmunds 提出了较适合于计算机实现的多变量参数最优化设计算法 [10]，英国学者 Zakian 提出的不等式方法 [34, 35] 也不失为一种较有发展前途的设计方法。

前面提及的各种方法有一个共同的特点，就是它们都根据系统的传递函数来进行设计，所以这些方法又常常统称为频域设计方法。本章将系统地介绍各种频域 CACSD 方法，而第 7 章中将介绍基于状态方程的时域设计方法。

6.2 多变量系统的频域设计方法

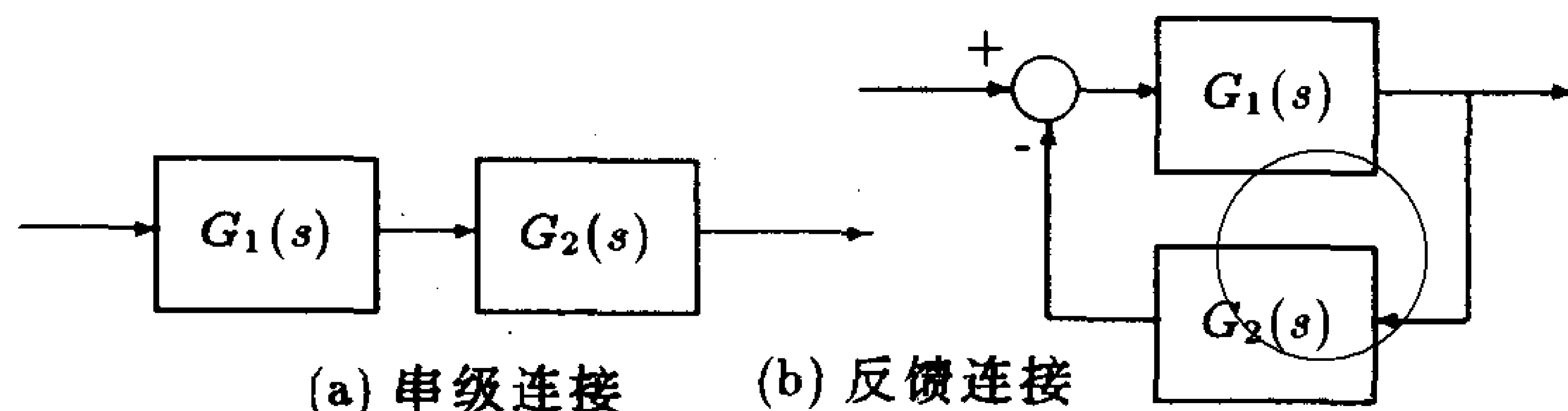
6.2.1 多变量系统的数学模型及标准型表示

第 4 章中介绍了控制系统的数学模型问题，并将着重点放在单变量系统的研究上，在本节中将探讨多变量系统的数学模型表示形式，并给出闭环系统的描述方法。多变量



系统其实也不外乎传递函数模型与状态方程模型两种描述方法，其中的状态方程模型不论单变量还是多变量的其描述方式是一致的，所以在状态方程模型下仍可以沿用第 4 章中的内容来表示多变量系统，如果想采用传递函数的形式来描述一个多变量系统，则将和传统的单变量模型有很大的区别，首先，多变量系统的传递函数不再单单是分子分母多项式的问题，而必须表示成传递函数矩阵的形式。所谓传递函数矩阵是指矩阵的各个元素都是单一的传递函数，则其中 $g_{ij}(s)$ 表示第 i 输入对第 j 输出的传递函数。另外在不同的连接结构下系统总的传递函数矩阵求法是不同的。考虑图 6-1 (a) 所示的串级连接结构，其总的传递函数矩阵可以写成

$$G(s) = G_2(s)G_1(s),\ \text{切记一般不可写成}\ G_1(s)G_2(s) \tag{6.2.1}$$



(a) 串级连接　(b) 反馈连接

图6-1　多变量系统的连接结构

在单变量系统连接下就不存在这样的交换问题。再考虑图 6-1 (b) 所示的典型反馈控制结构，这时总体传递函数矩阵可以写成

$$G(s) = G_1(s)[I + G_1(s)G_2(s)]^{-1} = [I + G_2(s)G_1(s)]^{-1}G_1(s) \tag{6.2.2}$$

而不是像单变量系统那样简单地求一下倒数就可以了，所以多变量系统处理起来更麻烦。多变量系统的传递函数矩阵也有各种标准型表示，其中最常用的是 Smith-McMillan 标准型，可以将 $G(s)$ 分解为

$$G(s) = L(s)M(s)R^{-1}(s) \tag{6.2.3}$$

其中 $L(s)$ 和 $R(s)$ 为幺模矩阵 (unimodular matrix), 即它们的行列式恒等于非 0 常数，这两个矩阵为初等变换矩阵，标准型 $M(s)$ 阵可以写成

$$M(s) = \mathrm{diag}\left\{\frac{\gamma_1(s)}{\psi_1(s)}, \frac{\gamma_2(s)}{\psi_2(s)}, \cdots, \frac{\gamma_r(s)}{\psi_r(s)}, 0, \cdots\right\} \tag{6.2.4}$$

其中 $\gamma_i(s)$ 可以整除 $\gamma_{i+1}(s)$, 记作 $\gamma_i(s) \mid \gamma_{i+1}(s)$, 且有 $\psi_{i+1}(s) \mid \psi_i(s)$ 。求取 Smith-McMillan 标准型的方法有些类似于求解线性代数方程组的 Gauss 消元法，首先取出整个传递函数矩阵的公分母，令之为 $\psi_1(s)$, 并寻找最小的分子 $\gamma_1(s)$, 再通过变换将之换到第 1 行第 1 列的位置上，然后类似于 Gauss 消去法的原理将第 1 行第 1 列的其它所有元素均变换为 0 。对右下角剩余子矩阵重复上面过程，将最终获得系统的 Smith-McMillan 标准型矩阵 $M(s)$ 。在 Smith-McMillan 标准型中 $\psi_1(s)$ 多项式的阶次又称为系统的 McMillan 阶次。

例 6.1　假设系统的传递函数矩阵为

$$G(s)=\begin{bmatrix}\dfrac{1}{s^2+3s+2} & \dfrac{-1}{s^2+3s+2}\\ \dfrac{s^2+s-4}{s^2+3s+2} & \dfrac{2s^2-s-8}{s^2+3s+2}\\ \dfrac{s-2}{s+1} & \dfrac{2s-4}{s+1}\end{bmatrix}$$

显然此系统的公分母为 $\psi_1(s)=s^2+3s+2=(s+1)(s+2)$，而第 1 行第 1 列元素为最小分子形式，所以并不需要对之进行换行和换列处理，用户可以通过下面的各个步骤得出变换后的 $G(s)$

$$G(s)\to\begin{bmatrix}\dfrac{1}{s^2+3s+2} & 0\\ \dfrac{s^2+s-4}{s^2+3s+2} & \dfrac{3s^2-12}{s^2+3s+2}\\ \dfrac{s-2}{s+1} & \dfrac{3s-6}{s+1}\end{bmatrix}\to\begin{bmatrix}\dfrac{1}{s^2+3s+2} & 0\\ 0 & \dfrac{3s^2-12}{s^2+3s+2}\\ 0 & \dfrac{3s-6}{s+1}\end{bmatrix}\to\begin{bmatrix}\dfrac{1}{s^2+3s+2} & 0\\ 0 & \dfrac{s-2}{s+1}\\ 0 & 0\end{bmatrix}$$

在前面的变换中第一步先将第 1 列的元素加到第 2 列上，以使得右上角元素为 0，第二步将第 1 行乘以 $-(s^2+s-4)$ 加于第 2 行，并将第 1 行乘以 $-(s-2)$ 加于第 3 行，以消去第 1 列的其余元素。第三步变换之前先将 $\hat{g}_{22}(s)$ 化简成 $3(s-2)/(s+2)$，然后由第 3 行元素减去第 2 行元素，以消去右下角元素，再将第 2 行乘以 1/3，最后将得出 Smith-McMillan 标准型 $M(s)=\mathrm{diag}\{1/(s^2+3s+2),(s-2)/(s+1)\}$。

如果已知多变量系统的状态方程模型，则可以由 eig(A) 函数来直接求出系统的极点，系统的传输零点 (transmission zeros) 可以由控制系统工具箱中给出的 tzero() 函数来直接求出，该函数的调用格式为

```
Z = tzero(A,B,C,D)
```

可见这一函数的调用是很直观的，由给定系统的状态方程模型 (A,B,C,D) 各个矩阵可以直接获得系统的传输零点 Z。

例 6.2　假设一个多变量系统的状态方程模型为

$$\dot{x}=\begin{bmatrix}-2&-6&3&-7&6\\0&-5&4&-4&8\\0&2&0&2&-2\\0&6&-3&5&-6\\0&-2&2&-2&5\end{bmatrix}x+\begin{bmatrix}-2&7\\-8&-5\\-3&0\\1&5\\-8&0\end{bmatrix}u,\quad y=\begin{bmatrix}0&-1&2&-1&-1\\1&1&1&0&-1\\0&3&-2&3&-1\end{bmatrix}x$$

可以由下面的命令容易地输入此系统模型，并求出系统的极点和传输零点

```
>> A=[-2,-6,3,-7,6; 0,-5,4,-4,8; 0,2,0,2,-2; 0,6,-3,5,-6; 0,-2,2,-2,5];
>> B=[-2,7; -8,-5; -3,0; 1,5; -8,0];
>> C=[0,-1,2,-1,-1; 1,1,1,0,-1; 0,3,-2,3,-1]; D=zeros(3,2);
>> P=eig(A)
P=  -2.0000
    -1.0000
     3.0000
     1.0000
     2.0000
```

```
>>  Z=tzero(A,B,C,D)
Z = -3.0000
     4.0000
```

6.2.2 多变量系统的频率响应

基于频域响应的设计方法在设计单变量系统时是最常用的方法，其主要原因是在设计过程中可以产生很多可视的图形，用户可以通过观察图形来决定对控制器参数的调节。直至六十年代末期英国学者 Harold Rosenbrock, Alistair MacFarlane 等开始研究多变量系统的频域设计方法，并取得了一系列引人注目的成就，在控制界称其研究为英国学派 (British School, 而在同一时间美国学者更侧重于基于状态空间的设计研究)。

英国剑桥大学学者 Boyel 和 Maciejowski 等推出的多变量频域设计 (MFD) 工具箱 [9] 很适合于求解频域设计问题，它提供了一系列函数来对频域模型进行分析，在介绍使用 MFD 工具箱之前首先介绍多变量系统的 MATLAB 表示方法。

如果已知多变量系统的状态方程模型，则当然可以由它求取各种频率响应。如果已知系统的传递函数矩阵，则模型的输入就不那么直观了。要输入这样的模型首先要求出系统的公分母，然后再求出在此公分母下传递函数矩阵的各个元素的分子多项式，最后表示出系统的规范传递函数矩阵模型。

例 6.3　考虑下面给出的 2 输入 2 输出传递函数矩阵

$$G(s)=\begin{bmatrix}\dfrac{s+4}{(s+1)(s+5)} & \dfrac{1}{5s+1}\\ \dfrac{s+1}{s^2+10s+100} & \dfrac{2}{2s+1}\end{bmatrix}$$

由上面的模型可以很容易地求出系统的公分母为

$$D(s)=10s^6+167s^5+1763s^4+7671s^3+9715s^2+4150s+500$$

这样就可以改写系统的传递函数矩阵的分子，最终可以写成

$$\begin{bmatrix}10s^5+147s^4+1499s^3+4994s^2+2940s+400 & 2s^5+33s^4+346s^3+1465s^2+1650s+500\\ 10s^5+77s^4+160s^3+134s^2+46s+5 & 10s^5+162s^4+1682s^3+6830s^2+6300s+1000\end{bmatrix}$$

有了上面的规范模型表示就可以直接写出系统的 MATLAB 表示了

```
>>  num = [0 10 147 1499 4994 2940 400, 0  2  33  346 1465 1650  500
           0 10  77  160  134   46   5, 0 10 162 1682 6830 6300 1000];
>>  den = [10 167 1763 7671 9715 4150 500];
```

若给定多变量系统的传递函数矩阵模型，也可以容易地获得相应的状态方程模型，多变量系统状态方程实现的方法并不像单变量系统那么简单、直观，假设系统的传递函数阵 $G(s)$ 可以写成 $G(s)=[g_1(s),g_2(s),\cdots,g_l(s)]$，且已知该系统有 l 个输入量，m 个输出量，其中 $g_i(s)$ 为列向量，且记 $g_i(s)=n_i(s)/d_i(s)+\delta_i$，则 $d_i(s)$ 为该列的公分母，其首一化的形式可以写成

$$d_i(s)=s^{k_i}+d_i^1s^{k_i-1}+d_i^2s^{k_i-2}+\cdots+d_i^{k_i} \tag{6.2.5}$$



$n_i(s)$ 为多项式列向量，每个多项式的阶次均小于 k_i, 且第 j 个子多项式可以表示成

$$v_{ji}(s) = v_{ji}^1 s^{k_i-1} + v_{ji}^2 s^{k_i-2} + \cdots + v_{ji}^{k_i} \tag{6.2.6}$$

且 δ_i 为常数列向量 (可以为 0)。这样子系统 $g_i(s)$ 的实现可以表示成

$$A_i = \begin{bmatrix} -d_i^1 & -d_i^2 & \cdots & -d_i^{k_i-1} & -d_i^{k_i} \\ 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix},\ B_i = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},\ C_i = \begin{bmatrix} v_{1i}^1 & \cdots & v_{1i}^{k_i} \\ v_{2i}^1 & \cdots & v_{2i}^{k_i} \\ \vdots & \ddots & \vdots \\ v_{mi}^1 & \cdots & v_{ni}^{k_i} \end{bmatrix} \tag{6.2.7}$$

这时整个系统的状态方程实现为

$$A = \operatorname{diag}(A_1, \cdots, A_l),\ B = \operatorname{diag}(B_1, \cdots, B_l),\ C = [C_1, \cdots, C_l],\ D = [\delta_1, \cdots, \delta_l] \tag{6.2.8}$$

这样得出的状态方程模型一定是可控的，但不一定可观测，一般情况下这样的转换会引入很多不必要的状态变量，所以得出此状态方程实现之后最好作最小实现处理，以消除不可观测的状态，可以证明，系统最小实现的阶次等于其 McMillan 阶次。MFD 工具箱中给出了一个从传递函数矩阵求取状态方程的函数 mvtf2ss(), 其调用格式为

```
[A, B, C, D]=mvtf2ss(num,dencom)
```

其中 num 和 dencom 和前面的定义是一致的，返回的 A, B, C, D 为相应的状态方程模型，其中这样得出的状态方程为最小实现的结果。

考虑前面给出的多变量传递函数模型，其相应的状态方程模型为

```
>>  [A,B,C,D]=mvtf2ss(num,den)
A = 1.0e+003 *
     -0.0034   -0.0138    0.0015    0.5022    1.1043    0.2755
     -0.0002   -0.0046    0.0005    0.1709    0.3752    0.0943
      0.0003    0.0001   -0.0002    0.0022    0.0052   -0.0006
      0.0001   -0.0006   -0.0004   -0.0071   -0.0163   -0.0043
      0.0000    0.0000    0.0000    0.0002   -0.0006    0.0001
      0.0000    0.0000    0.0000   -0.0005   -0.0013   -0.0008
B = -0.9436    0.0001
    -0.3209   -0.0003
    -0.0041    0.0010
     0.0138    0.0001
     0.0010   -0.0003
     0.0013    0.0010
C =  0.0000    0.0000    0.0000    0.0000  530.7401  361.7095
     0.0000    0.0000    0.0000    0.0000 -207.0603  927.1391
D =  0      0
     0      0
```

若已知系统的状态方程模型 (A,B,C,D)，当然可以和单变量系统一样由下式获得相应的传递函数矩阵模型

$$G(s)=C(sI-A)^{-1}B+D=\frac{C\mathrm{adj}(sI-A)B}{\det(sI-A)}+D \tag{6.2.9}$$

也可以调用 ss2tf(),但 MFD 工具箱中提供了更简单的函数 mvss2tf() 函数，该函数的调用格式为

```
[num,dencom]=mvss2tf(A,B,C,D)
```

其中参数的定义和前面是完全一致的。

重新考虑上面得出的状态方程结果，调用 mvss2tf() 函数可以得出传递函数矩阵的结果

```
>>  [num1,den1]=mvss2tf(A,B,C,D)
num1 =  Columns 1 through 7
         0     1.0000    14.7000   149.9000   499.4000   294.0000    40.0000
         0     1.0000     7.7000    16.0000    13.4000     4.6000     0.5000
          Columns 8 through 14
         0     0.2000     3.3000    34.6000   146.5000   165.0000    50.0000
         0     1.0000    16.2000   168.2000   683.0000   630.0000   100.0000
den1 = 1.0000    16.7000   176.3000   767.1000   971.5000   415.0000    50.0000
>>  [norm(num/10-num1) norm(den/10-den1)]
ans =   1.0e-010 *
         0.1088     0.0426
```

注意这里原系统的分母多项式分别除以 10 是因为原系统 den 并不是首一表示，而由 mvss2tf() 函数得出的是首一的，所以比较它们是应该作适当的处理。由此结果可见经过反变换也基本可以得出原传递函数矩阵模型。

若给出了系统的传递函数矩阵，则可以调用 mv2fr() 函数来求取系统的频域响应数值，该函数的调用格式为

```
mf=mv2fr(num,den,w)
```

或

```
mf=mv2fr(A,B,C,D,w);
```

其中 w 为预先选定的频率向量， den 为系统的公分母系数向量，而 num 为系统在公分母 den 下的传递函数矩阵分子系数。调用这一函数返回的即是系统的频域响应数值构成的以复数形式表示的矩阵，该矩阵是由在每个频率点处的频率响应矩阵罗列而成的。在后一种调用格式下需要给出系统的状态方程模型 (A,B,C,D)。

当然，若想求出在第 iu 输入下的频率响应数据，则可以由下面格式调用 mv2fr() 函数

```
mf=mv2fr(A,B,C,D,w,iu);
```

或

```
mf=mv2fr(num,den,w,iu)
```

其实上面的调用格式还是很直观的。如果不依赖于多变量系统工具箱，也是很容易求出系统的频率响应的。

如果已知系统的传递函数矩阵模型(并不要求事先得出公分母形式)，则可以提供下面的程序 newmv2fr.m 来求出和 mv2fr() 同样的结果。

```
mf=[];
for i=1:q
   mf1=[];
   for j=1:p
      eval(['mf1=[mf1 getfreqd(num' int2str(i) int2str(j), ...
         ', den' int2str(i), int2str(j), ', w, dly(',...
         int2str(i), ',' int2str(j) '))];']);
   end
   mf(i:q:length(w)*q,:)=mf1;
end
```

其中系统的传递函数矩阵中各个元素的分子和分母将存放在num11，den11，num12，den12，···，numpq，denpq 中，这里 p 和 q 分别对应于输入和输出的个数。除此之外，还应该提供系统的时间延迟矩阵 dly 。这里调用了用户自定义函数 getfreqd(), 它可以用来求取带有纯时间延迟的单变量系统的频率响应数据。

```
function h=getfreqd(num,den,w,dly)
if nargin==3, dly=0; end
[mag,phase]=bode(num,den,w); phase=phase*pi/180-dly*w';
x=mag.*cos(phase); y=mag.*sin(phase); h=x+sqrt(-1)*y;
```

要想求解出例 6.3 中给出的多变量系统的频率响应数据，首先应该按照下面的方法给出系统的传递函数矩阵模型和频率点向量，然后画出图形。

```
>> num11=[1 4]; den11=conv([1 1],[1,5]); num12=1; den12=[5 1];
>> num21=[1,1]; den21=[1,10,100]; num22=[2]; den22=[2,1]; dly=zeros(2,2);
>> p=2;  q=2; w=logspace(-2,2);
>> newmv2fr; mimonyq(mf,2)
```

前面程序 newmv2fr.m 的调用将得出和 mv2fr() 函数同样的结果，但此方法的优点是它不需要得出具有公分母的传递函数矩阵的形式。有了频率响应数据之后，则可以利用下面的 MATLAB 命令来绘制出系统的频率响应曲线来。我们可以编写下面的函数 mimonyq.m 来绘制出多变量 Nyquist 曲线

```
function mimonyq(mf,q,gr)
if (nargin==2), gr=0; end
[imax,p]=size(mf); ii=1:q:imax;
for i=1:p*q,
   i0=floor((i-1)/q); j0=i-i0*q;
   subplot(p,q,i); plot(real(mf(ii+i0,j0)),imag(mf(ii+i0,j0)));
   if (gr==1) grid; end
end
```

这时将得出图 6-2 所示的多变量 Nyquist 图形。

当然还可以使用 MFD 工具箱来绘制这些曲线

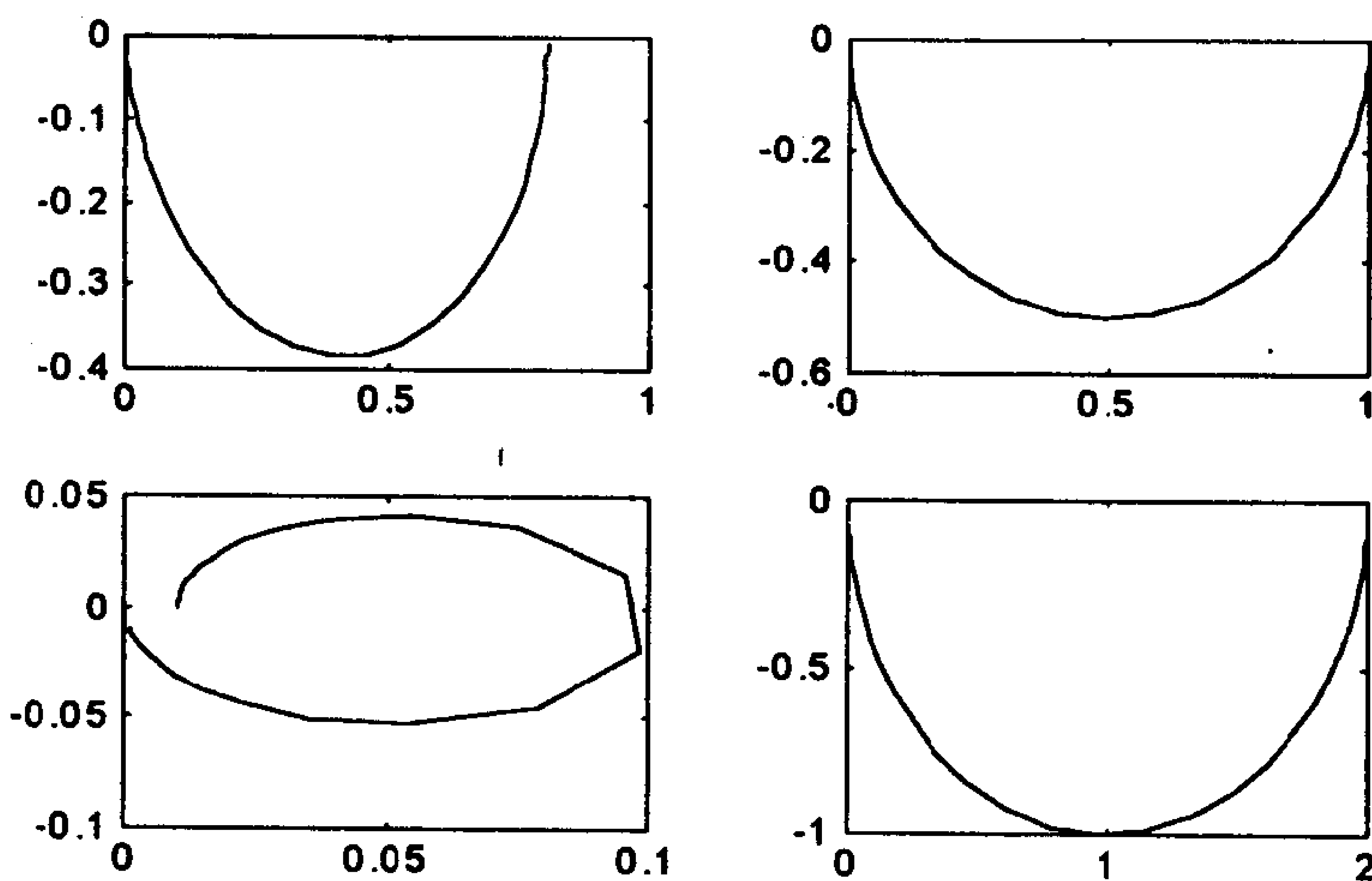


图6-2　系统的频率响应曲线

```
>> subplot(221); plotnyq(fget(w, mf, [1 1]));
>> subplot(222); plotnyq(fget(w, mf, [1 2]));
>> subplot(223); plotnyq(fget(w, mf, [2 1]));
>> subplot(224); plotnyq(fget(w, mf, [2 2]));
```

注意这里调用了 fget() 函数和 plotnqy() 函数来从得出的矩阵中获取数据，并绘制相应的 Nyquist 图形，其中 fget() 函数的调用格式如下

```
mv=fget(w, F, INDEX)
```

式中 w 为频率点构成的向量，F 为按前面方法得出的多变量频率响应数据的复数矩阵，INDEX 为输入输出对构成的指针表示，例如若想从 F 矩阵中提取第 i 输入信号对第 j 信号的频率响应数据，则可以将指针变量 INDEX 定义为 $[i, j]$，这时相应的频率响应数据将返回给 mv 矩阵。由此可见，可以容易地由得出的频率响应数据矩阵提取出所需的单变量子系统频率响应数据来，有了该频率响应数据，还可以采用 MFD 工具箱中提供的 plotnyq() 函数来绘制出相应的 Nyquist 曲线来，该函数的调用格式为

```
plotnyq(mv, 线型选项)    或    plotnyq(r, i, 线型选项)
```

其中 mv 为得出的单变量系统频率响应数据构成的列向量，线型选择和第 2 章中介绍的 plot() 函数的线型定义是一致的，在这里就不再另行叙述了。若给出了单变量系统频率响应数据的实部 r 和虚部 i 后，则可以采用第二种格式来绘制 Nyquist 图形。这两组命令得出的图形是一致的，请参见图 6-2 中给出的结果。在以后的叙述中将基于 MFD 工具箱中的函数来获取多变量系统的频率响应，而不再另行给出不使用该工具箱的相应命令了，用户当然可以参照以上的叙述来构造出不依赖于 MFD 工具箱的实现方法。

在多变量系统的分析中，人们往往更希望能绘制出系统的逆 Nyquist 图，所谓逆 Nyquist 数据 (inverse Nyquist array, 简称 INA) 即首先要求出原系统传递函数矩阵在各个频率点处的逆矩阵构成的新频率响应矩阵，记作

$$\widehat{G}(s)=\begin{bmatrix}\hat{g}_{11}(s) & \hat{g}_{12}(s) & \cdots & \hat{g}_{1p}(s)\\ \hat{g}_{21}(s) & \hat{g}_{22}(s) & \cdots & \hat{g}_{2p}(s)\\ \vdots & \vdots & \ddots & \vdots\\ \hat{g}_{q1}(s) & \hat{g}_{q2}(s) & \cdots & \hat{g}_{qp}(s)\end{bmatrix} \tag{6.2.10}$$

事实上，若已知系统的频率响应数据 (这样的数据又称为直接 Nyquist 数据， direct Nyquist array, 简称 DNA)，则对每组数据求取逆矩阵就可以直接获得原系统的逆 Nyquist 响应数据。 MFD 工具箱提供了一个由 DNA 数据求取 INA 数据的函数 finv()，该函数的调用格式为

```
fout=finv(w, mv)
```

式中 w 为频率点向量， mv 为得出的 DNA 数据，这一函数的返回结果 fout 为原系统的逆 Nyquist 数据。其实用户自己用 MATLAB 实现 finv() 函数的功能并不困难，其主体程序可以通过下面的命令再现

```
function iG=finv(w,G)
[n,m]=size(G); p=length(w); iG=[];
if (n/m~=p) disp('Error: not square system'); iG=[]; return; end
for i=1:p, iG=[iG; inv(G(m*(i-1)+1:m*i,:))]; end
```

这样，原系统的逆 Nyquist 图形也同样可以采用前面介绍的 plotnyq() 绘制出来。

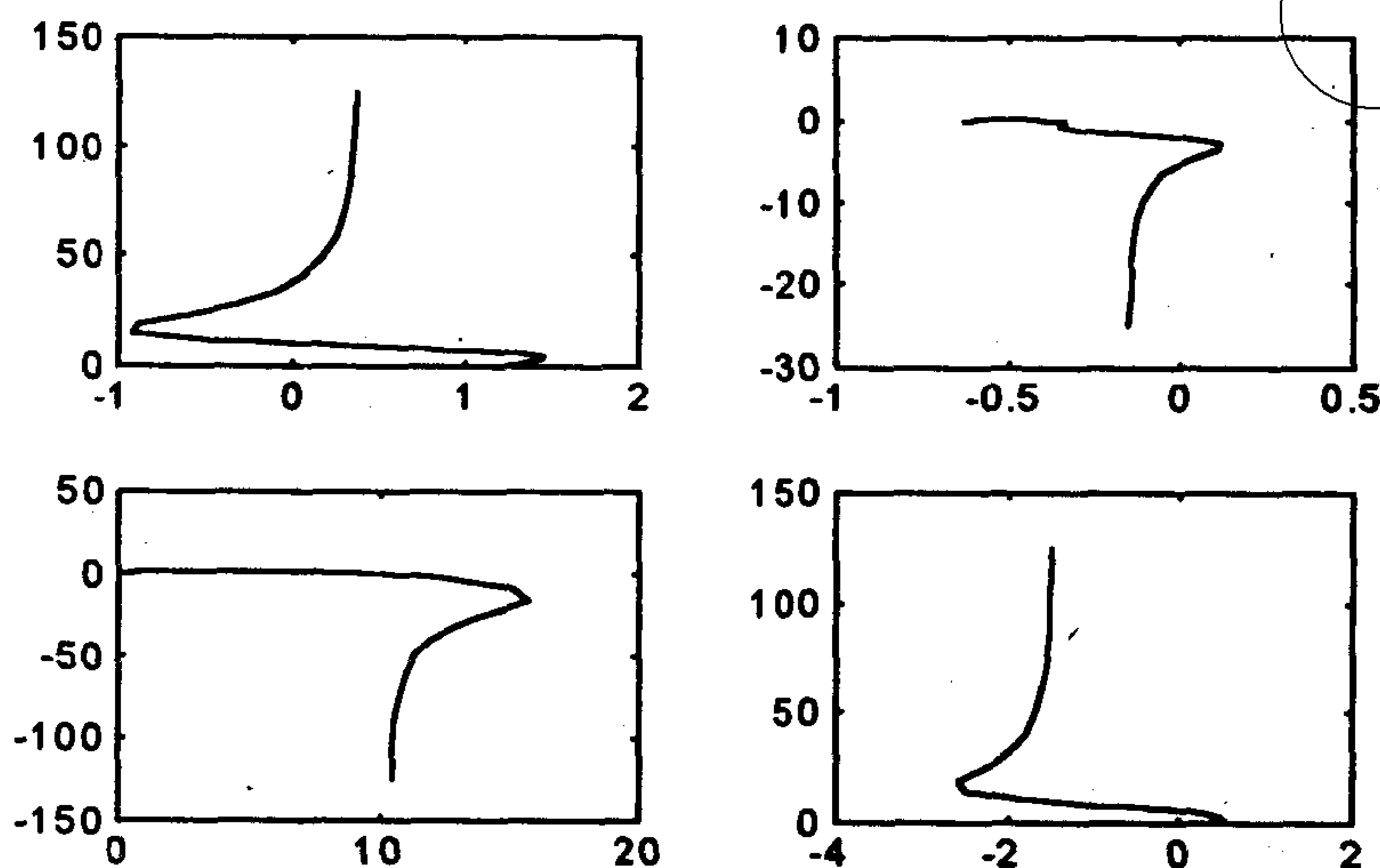


图6-3 逆 Nyquist 曲线

考虑前面的例题，若采用下面的命令则可以容易地绘制出系统的逆 Nyquist 曲线，如图 6-3 所示。

```
>> ifg = finv(w, mf);
>> subplot(221); plotnyq(fget(w, ifg, [1 1]));
>> subplot(222); plotnyq(fget(w, ifg, [1 2]));
>> subplot(223); plotnyq(fget(w, ifg, [2 1]));
>> subplot(224); plotnyq(fget(w, ifg, [2 2]));
```

这一效果由前面给出的 mimonyq() 函数的调用也可以直接完成

```
>> mimonyq(ifg,2,1)
```

由此可见，只要获得了按照 MFD 工具箱规则写出的频率响应数据矩阵，无论它是 DNA 还是 INA，都可以使用 mimonyq() 函数将它绘制出来。

6.2.3 对角占优系统与伪对角化

在多变量系统的频域分析中，经常要判断传递函数矩阵是否为对角占优矩阵 (diagonal dominance matrix)，即判断各个输入与输出之间的耦合情况，如果系统是对角占优型传递函数矩阵，则可以对各个回路进行单独地设计，而不会影响其它的回路。对角占优性的判断是依靠矩阵对角元素特征值范围判定的 Gershgorin 圆来完成的，对每一个频率点 ω 来说，多变量系统的频率响应数据将构成一个复数矩阵

$$\begin{bmatrix} \times & \cdots & c_{1k} & \cdots & \times \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ c_{k1} & \cdots & c_{kk} & \cdots & c_{kn} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \times & \cdots & c_{nk} & \cdots & \times \end{bmatrix} \tag{6.2.11}$$

对这一复数矩阵，由 Gershgorin 定理[29] 可知，其特征值 λ 满足下面的不等式

$$|\lambda - c_{kk}| \le \sum_{j\neq k} |c_{kj}|, \text{ 且 } |\lambda - c_{kk}| \le \sum_{j\neq k} |c_{jk}| \tag{6.2.12}$$

换句话说，该矩阵的特征值位于以 c_{kk} 为圆心，以不等式右面的表达式为半径的圆内，而该圆又称为 Gershgorin 圆。上面两个不等式表示的关系分别称为列 Gershgorin 圆和行 Gershgorin 圆。对于频率响应的所有数据来说，将由一系列 Gershgorin 圆构成一个 Gershgorin 带，若各个对角元素的 Gershgorin 带均不包含圆心，则称原系统为对角占优的。

在一族频率点处的 Gershgorin 圆的坐标点分别可以由 MFD 工具箱提供的 fcgersh() 函数和 frgersh() 函数来求得，这些函数的调用格式为

```
circles=fcgersh(W,F,i)    或    circles=frgersh(W,F,i)
```

这两个函数将分别返回列 Gershgorin 带和行 Gershgorin 带在各个频率点处的圆坐标值 circles，有了这些值之后，可以利用下面的各个语句在原来的 Nyquist 曲线或逆

Nyquist 曲线上容易地叠印出 Gershgorin 圆来，用户可以藉此判断当前的系统是否为对角占优的。

根据前面的叙述，对 mimonyq.m 文件稍作修改，则可以写出下面的文件

```
function mimonyq(w, mf, q, gcr, gr, ax)
if (nargin==3) gr=0; gcr=0;
elseif (nargin==4), gr=0; end
[imax,p]=size(mf); ii=1:q:imax;
for i=1:p*q,
   i0=floor((i-1)/q); j0=i-i0*q;
   subplot(p,q,i); plot(real(mf(ii+i0,j0)),imag(mf(ii+i0,j0)));
   if (j0==i0+1)
      if (gcr~=0)
         if (nargin==6) axis(ax(j0,:)); end
         hold on
         if (gcr==1), circles=fcgersh(w,mf,j0);
         elseif (gcr==2), circles=frgersh(w,mf,j0);
         end
         plot(real(circles),imag(circles)), hold off
      end
   end
   if (gr==1) grid; end
end
```

在这一函数的调用中，若 gcr=1 则绘制列 Gershgorin 带，若 gcr=2 则绘制行 Gershgorin 带，否则将只绘制 Nyquist 曲线，而不绘制 Gershgorin 带。 gr 变量的定义和前面的是一致的，即若 gr=1，则在画图时将网格线绘制出来，否则将不绘制网格线。如果给出第 6 个参数 ax，则将按照 $m \times 4$ 矩阵 ax 中的相应行的参数来设定对角图形的坐标。

重新分析例 6.3 中给出的系统，若得出系统的逆 Nyquist 频率响应矩阵 ifg，这时若想绘制出系统带有列 Gershgorin 带的逆 Nyquist 曲线，则可以给出下面的命令

```
>> mimonyq(w,ifg,2,1,0);
```

这时将立即绘制出叠印有列 Gershgorin 带的逆 Nyquist 曲线来，如图 6-4 所示，从该图可以看出，原系统并不是对角占优的系统，因为左上角元素的 Gershgorin 带并不能排除原点。

如果给定的传递函数矩阵不是对角占优的，则需要引入某种补偿方法将它化为对角占优的矩阵，然后可以不考虑各个输入信号之间的耦合，依照单变量系统的方法对各个输入进行单独地设计。 Nyquist 类方法最典型的控制框图如图 6-5 所示，其中 $K_p(s)$ 为预补偿矩阵，它使得 $G(s)K_p(s)$ 为对角占优矩阵，而 $K_d(s)$ 可以对得出的对角占优矩阵作动态的补偿，使之满足某些动态特性，达到设计的目的。

在多变量系统的设计中，求取 $K_p(s)$ 矩阵是关键的一步，它将决定最终设计的结果，在实际应用中往往将该矩阵设计成最简单的常数矩阵形式。用户可以根据自己的经验选中一个常数矩阵，该矩阵可以对系统的传递函数矩阵进行初等代数变换，使之成为对角

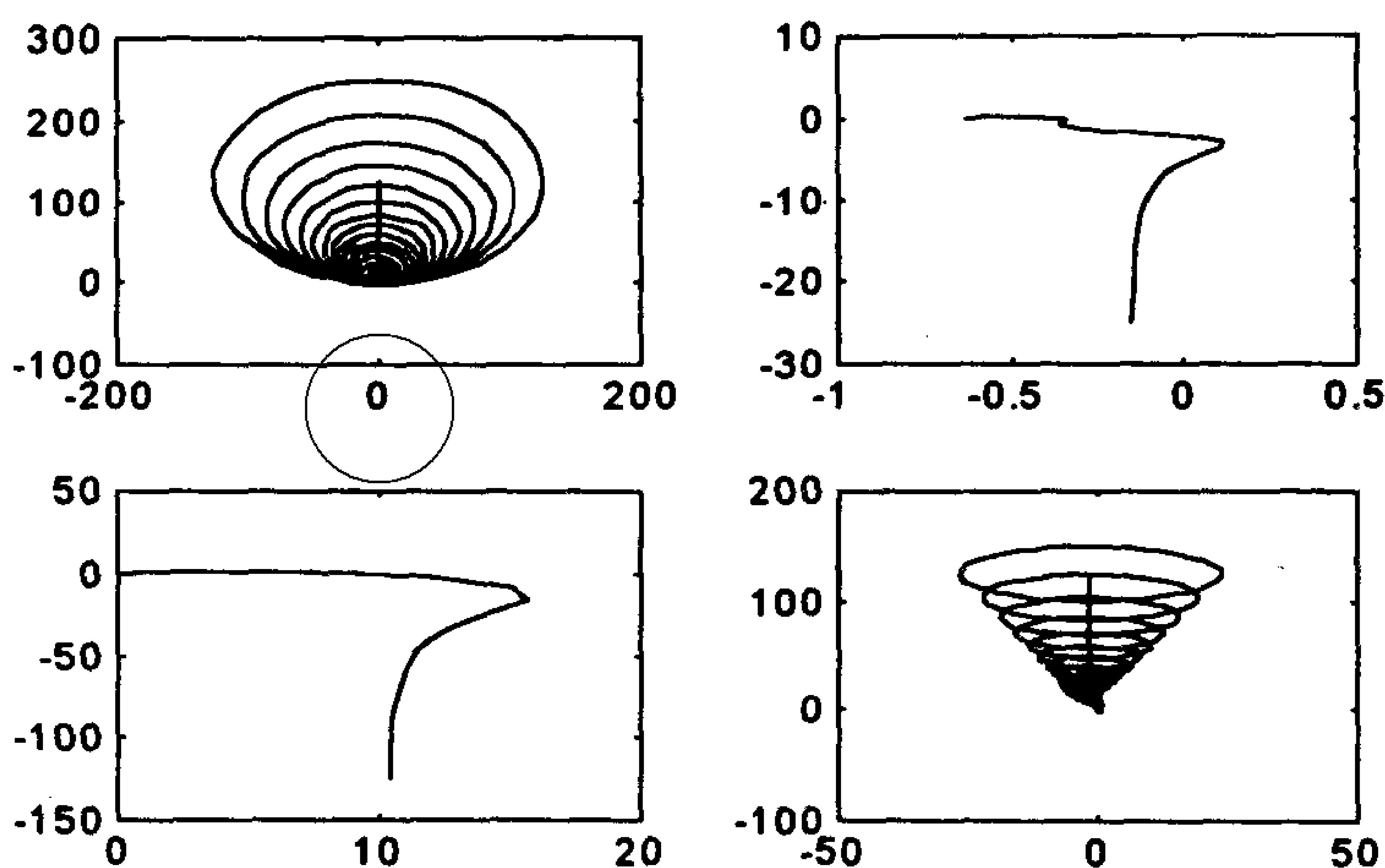


图6-4 带有 Gershgorin 带的逆 Nyquist 曲线

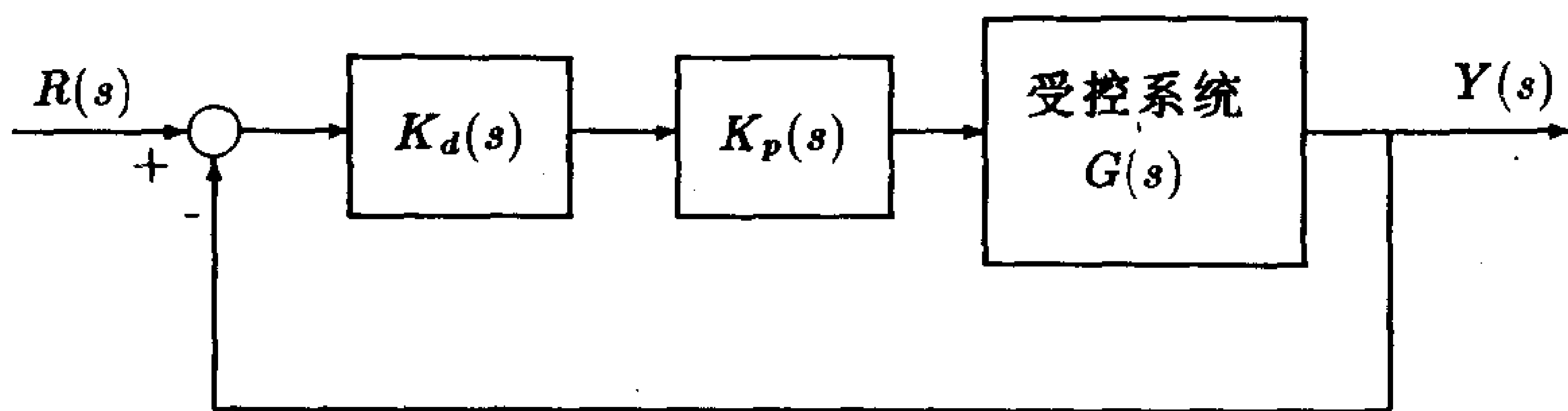


图6-5 典型多变量系统设计框图

占优的矩阵。选取 $K_p(s)$ 可以采用试凑的方法，一般可以将 $\widehat{K_p}(s)$ 选择为 $\widehat{K_p}(s) = G(0)$, 该矩阵至少可以使得 $G(s)K_p(s)$ 在频率为 0 时为单位矩阵，从而满足对角占优的要求。

采用试凑的方法毕竟不利于计算机辅助设计，所以很多学者提出不同的比较系统的方法来对传递函数矩阵进行对角占优化 [19], 下面将介绍一种最优化的方法来求取预补偿矩阵 K_p, 这一方法又称为伪对角化 (pseudo-diagonalisation) 方法。假设在 $j\omega_0$ 频率处的系统传递函数矩阵的逆 Nyquist 阵列表示为

$$\hat{g}_{ik}(j\omega_0) = \alpha_{ik} + j\beta_{ik},\ \ i,k = 1,\cdots,q \tag{6.2.13}$$

这里 q 为输出变量的个数，并假定系统的输入与输出个数相同。如果想获得一个最优的补偿矩阵 $\widehat{K_p}$, 则可以采用下面的步骤：

- 选择一个函数的频率点 $j\omega_0$, 求出系统的逆 Nyquist 阵列 $\hat{g}_{ik}(j\omega_0)$
- 对各个 γ 值 $(\gamma = 1,\cdots,m)$, 构成一个矩阵 A_γ

$$a_{il}^{\gamma} = \sum_{k=1\ 且 k\neq\gamma}^{q} \left[\alpha_{ik}\alpha_{lk} + \beta_{ik}\beta_{lk}\right],\ \ i,l = 1,\cdots,q \tag{6.2.14}$$

- 求取得出的 A_γ 矩阵的特征值与特征向量，并将最小特征值对应的特征向量记作 k_γ。
- 由上面的各个 γ 值得出的最小特征向量可以构成补偿矩阵 $\widehat{K}_p$

$$\widehat{K}_p = \left[k_1, k_2, \cdots, k_q\right]^T \tag{6.2.15}$$

依照上述的算法可以容易地由 MATLAB 编写出为对角化函数 pseudiag(), 该函数可以由给定频率点处的逆 Nyquist 数据 iG 来得出 $\widehat{K}_p$ 矩阵

```
function Kp=psuediag(iG)
A=real(iG); B=imag(iG);
[n,m]=size(iG); K=[];
for q=1:m
   for i=1:m
      for l=1:m
         ll=[1:q-1, q+1:m];
         Ap(i,l)=sum(A(i,ll).*A(l,ll)+B(i,ll).*B(l,ll));
   end,end
   [x,d]=eig(Ap); [xm,ii]=min(diag(d));
   K=[K; x(:, ii)'];
end
```

例 6.4 考虑下面的 4 输入 4 输出蒸汽锅炉温度控制模型[29]

$$G(s) = \begin{bmatrix} 1/(1+4s) & 0.7/(1+5s) & 0.3/(1+5s) & 0.2/(1+5s) \\ 0.6/(1+5s) & 1/(1+4s) & 0.4/(1+5s) & 0.35/(1+5s) \\ 0.35/(1+5s) & 0.4/(1+5s) & 1/(1+4s) & 0.6/(1+5s) \\ 0.2/(1+5s) & 0.3/(1+5s) & 0.7/(1+5s) & 1/(1+4s) \end{bmatrix}$$

可以输入该系统模型，并采用多变量工具箱中给出的函数绘制出系统的逆 Nyquist 曲线及行 Gershgorin 带

```
>> den=conv([5,1],[4,1]); d1=[0,4,1]; d2=[0,5,1];
>> num=[d2,0.7*d1,0.3*d1,0.2*d1; 0.6*d1, d2,0.4*d1,0.35*d1;
       0.35*d1,0.4*d1,d2,0.6*d1; 0.2*d1,0.3*d1,0.7*d1,d2];
>> w=logspace(-1,0,15); G=mv2fr(num,den,w); iG=finv(w,G);
>> for i=1:4
      subplot(2,2,i), plot([0,0],[-2,11],'-',[-4,8],[0,0],'-')
      axis([-4,8,-2,11]);hold on; C=frgersh(w,iG,i); plotnyq(C,'-');
      plotnyq(fget(w,iG,[i,i])); hold off;
   end
```

这时该系统带有行 Gershgorin 带的逆 Nyquist 图如图 6-6 所示，注意这里只显示了对角元素的逆 Nyquist 图。由该图可见原系统是对角占优的，但在低频处这种优势并不是很强，所以应该进一步补偿，来强化其对角优势。

首先可以考虑引入 $\widehat{K}_{p1} = G(0)$, 这时可以得出如图 6-7 所示的对角元素逆 Nyquist 曲线，可见其对角优势在很大程度上得到了强化。

```
>> Kp=num(:,3:3:12); iG1=[];
>> for i=1:length(w), iG1=[iG1; Kp*iG(4*(i-1)+1:4*i,:)]; end
>> for i=1:4
      subplot(2,2,i), plot([0,0],[-1,6],'-',[-2,4],[0,0],'-')
      axis([-2,4,-1,6]);hold on; C=frgersh(w,iG1,i);
      plotnyq(C,'-'); plotnyq(fget(w,iG1,[i,i])); hold off;
   end
```

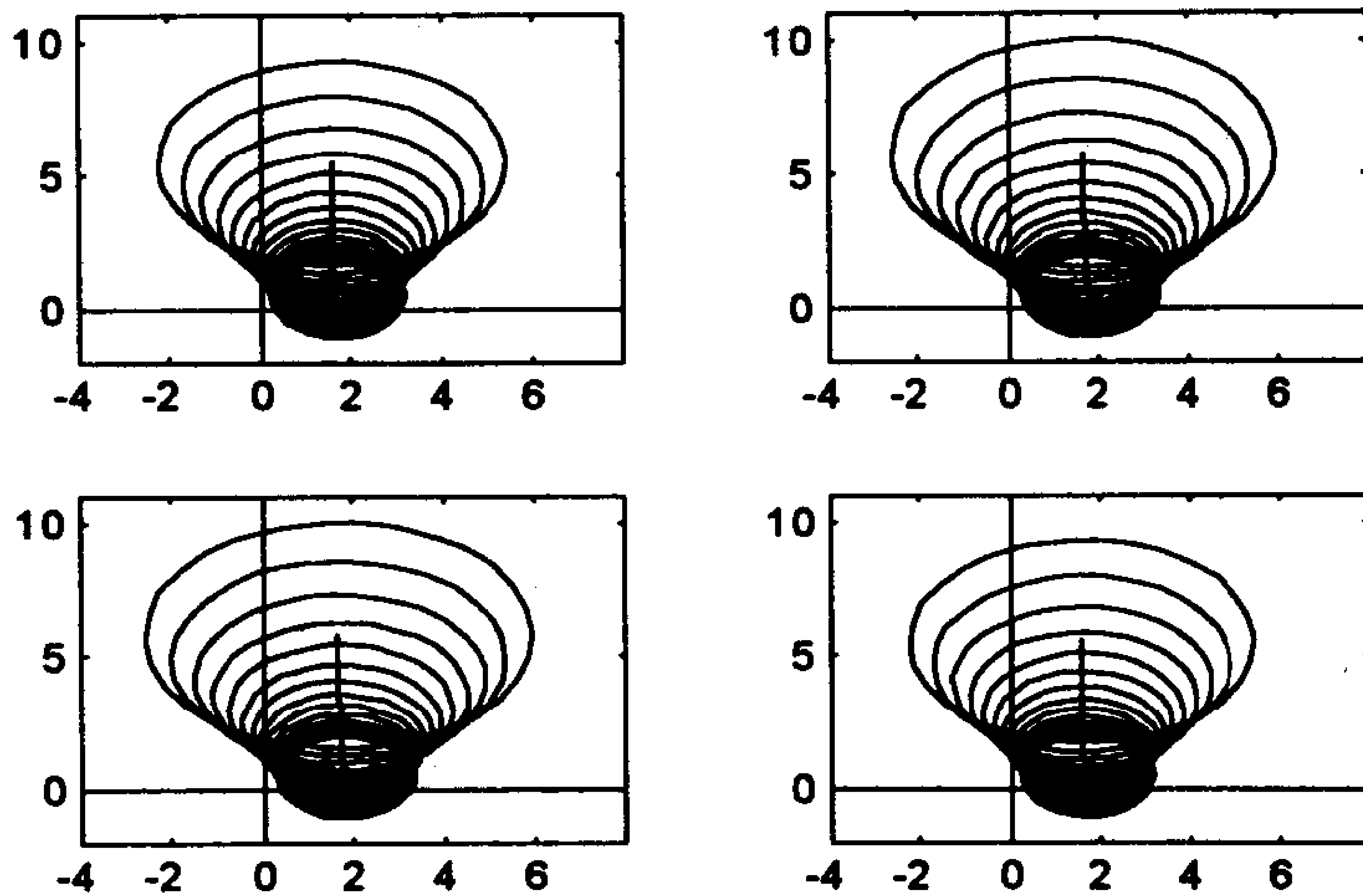


图6-6　蒸汽锅炉温度控制模型逆 Nyquist 图

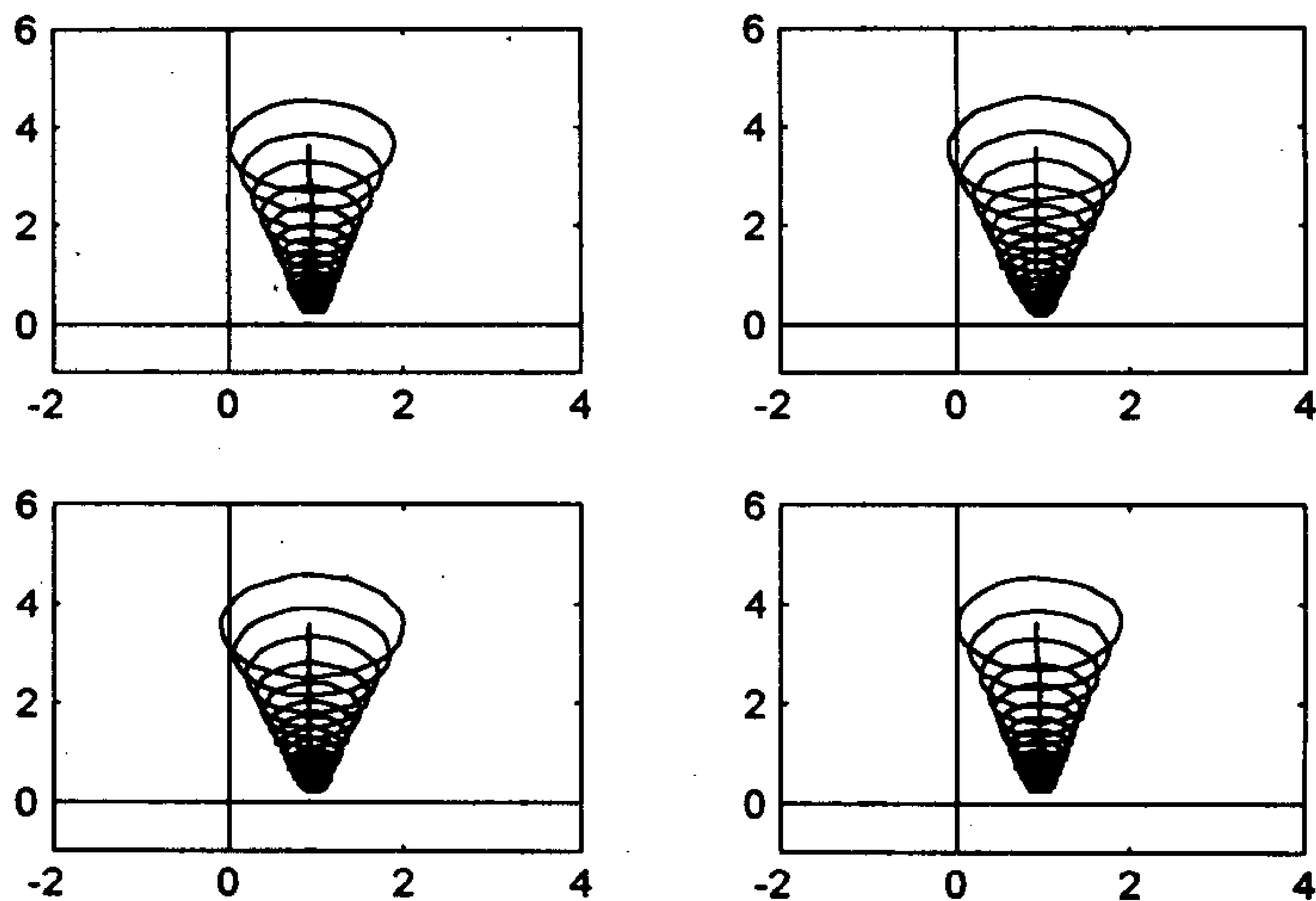


图6-7　选取 $\widehat{K}_p = G(0)$ 时逆 Nyquist 图

再考虑根据上面的 pseudiag() 函数对 $\omega_0 = 0.9$ 作伪对角处理，则可以得出系统的补偿矩阵为

```
>>  ww=0.9; Gw=mv2fr(num,den,ww); iGw=finv(ww,Gw); iG2=[]; Kp=pseudiag(iGw)
Kp =  0.8427    0.4781    0.2057    0.1376
     -0.4075   -0.8386   -0.2721   -0.2380
     -0.2380   -0.2721   -0.8386   -0.4075
     -0.1376   -0.2057   -0.4781   -0.8427
>>  for i=1:length(w), iG2=[iG2; Kp*iG(4*(i-1)+1:4*i,:)]; end
>>  for i=1:4
       subplot(2,2,i), plot([0,0],[-4,4],'-',[-1.5,1.5],[0,0],'-')
       axis([-1.5,1.5,-4,4]);hold on; C=frgersh(w,iG2,i);
       plotnyq(C,'-'); plotnyq(fget(w,iG2,[i,i])); hold off;
    end
```

这时补偿模型的逆 Nyquist 图如图 6-8 所示，由此可见，这样补偿后的系统已经趋于对角矩阵了，所以可以采用类似于单变量系统的方法去进行设计，而不必顾忌某一回路会影响其它回路。

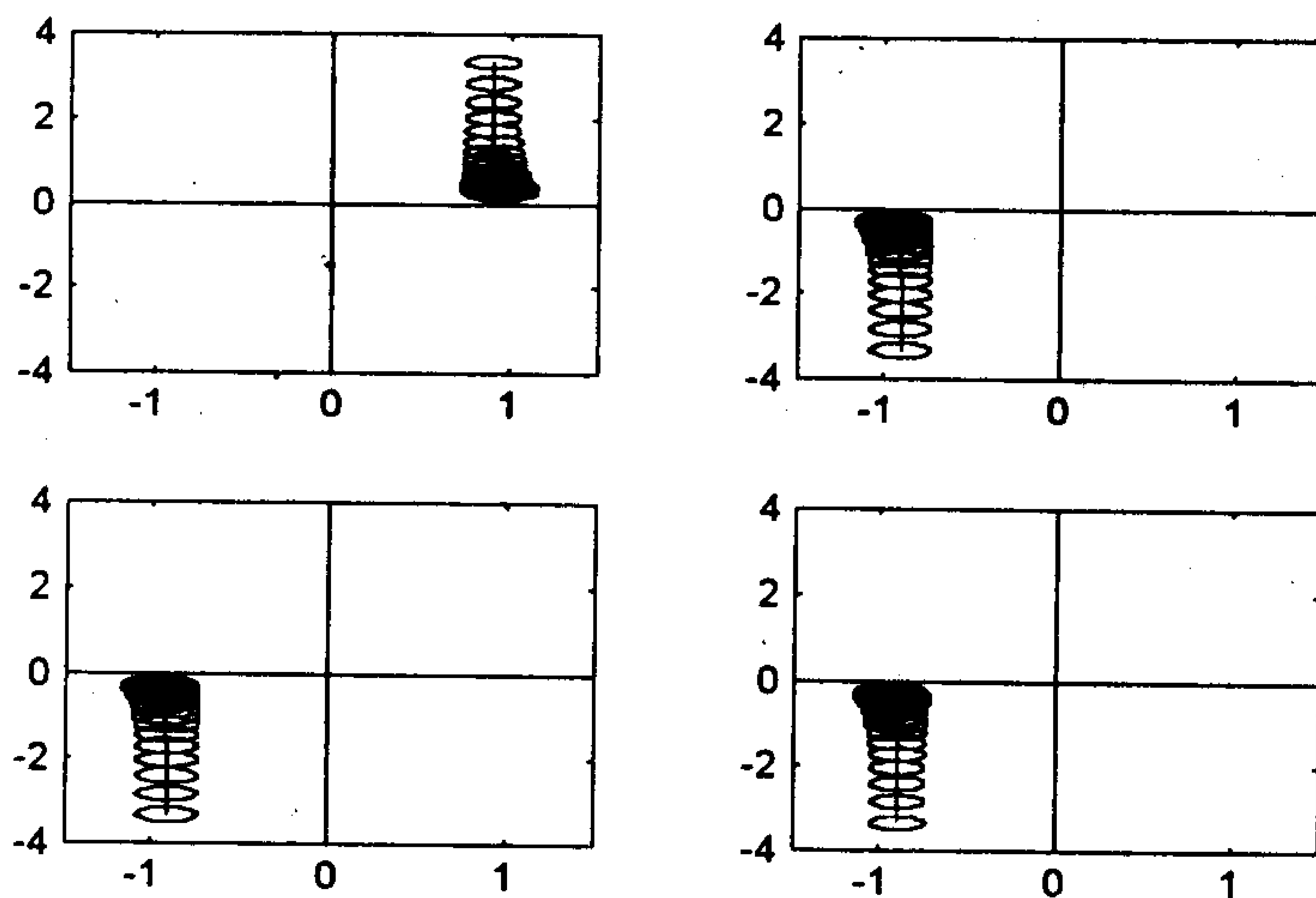


图 6-8　$\omega_0=0.9$ 时逆 Nyquist 图

当然前面介绍的伪对角化方法是基于某一频率的，而具体应该针对哪个频率去设计还应该通过试凑的方法来完成，下面将简介一种方法来对某个频率段进行加权来实现伪对角化，选择 N 个频率点 $\omega_1,\omega_2,\cdots,\omega_N$，并假设对第 r 个频率点引入加权系数 ψ_r，按照如下的方法构造 B 矩阵

$$b_{il}^{\gamma}=\sum_{r=1}^{N}\psi_r\left[\sum_{k=1\ 且k\neq\gamma}^{q}(\alpha_{ik}^{(r)}\alpha_{lk}^{(r)}+\beta_{ik}^{(r)}\beta_{lk}^{(r)})\right] \tag{6.2.16}$$

其中 $\alpha^{(r)}$ 和 $\beta^{(r)}$ 表示第 r 点处的 α 和 β 值，这样就可以仿照前面的方法来求取伪对角化矩阵 K_p 了。文献 [11] 中还仿照这一伪对角化的方法给出了构造动态补偿器 $K_p(s)$ 的方法，读者可以自己去查阅有关文献。

6.2.4 多变量系统的设计方法举例

值得指出的是，为设计简单方便起见，一般在使用多变量系统的 Nyquist 类设计方法时多采用逆 Nyquist 阵列的方法，下面不加证明地给出逆 Nyquist 设计方法的稳定性定理。

若 G 为 $m\times m$ 方阵且其逆 Nyquist 矩阵的第 (i,j) 元素为 $\hat{g}_{ij}(s)$，且存在 $K=\operatorname{diag}(k_1,k_2,\cdots,k_m)$，假定对所有可取的 s 均有

$$|\hat{g}_{ii}(s)+k_i|>\sum_{j\neq i}|\hat{g}_{ij}(s)| \tag{6.2.17}$$

设 $\widehat{G}(s)$ 的第 i 个 Gershgorin 带逆时针方向包含 $-k_i$ 点 N_i 次，则带有返回比矩阵 $-G(s)K$ 的负反馈矩阵是稳定的充要条件是 $\sum_i N_i=Z_o$，这里 Z_o 是 $G(s)$ 在右半平面的传输零点。

对典型的负反馈结构 (例如图 6-1(b) 所示的结构) 来说，所谓返回比 (return ratio) 矩阵 [21] 可以定义为 $-G_1(s)G_2(s)$ 或 $-G_2(s)G_1(s)$，亦即原反馈结构变成开路时的开环传递函数矩阵，其中对通路 $G_1(s)$ 来说，前者又称为输出端返回比矩阵，而后者称为输入端返回比矩阵，同样还可以定义出返回差 (return difference) 矩阵为 $I+G_1(s)G_2(s)$ 或 $I+G_2(s)G_1(s)$，且前者称为输出端返回差矩阵，而后者称为输入端返回差矩阵。

但逆 Nyquist 设计方法有一个最大的约束：原系统的传递函数矩阵为方阵，亦即系统的输入和输出个数是相同的，因为只有这样才能保证系统的逆 Nyquist 矩阵的存在。若系统传递函数矩阵不是方阵时则不能采用 Nyquist 方法，而必须采用直接 Nyquist (即 DNA 法) 方法来设计。

例 6.5 已知含有延迟的多变量系统传递函数矩阵为 [26]

$$G(s)=\begin{bmatrix}\dfrac{0.1134\mathrm{e}^{-0.72s}}{1.78s^2+4.48s+1} & \dfrac{0.924}{2.07s+1}\\ \dfrac{0.3378\mathrm{e}^{-0.3s}}{0.361s^2+1.09s+1} & \dfrac{-0.318\mathrm{e}^{-1.29s}}{2.93s+1}\end{bmatrix}$$

可以通过下面的方法输入系统的模型，并得出逆 Nyquist 图数据，从而绘制出原系统的逆 Nyquist 图，如图 6-9 所示。

```
>> num11=0.1134; num12=0.924; num21=0.3378; num22=-0.318;
>> den11=[1.78,4.48,1]; den12=[2.07,1]; den21=[0.361,1.09,1]; den22=[2.93,1];
>> dly=[0.72, 0; 0.3,1.29]; w=logspace(0,1); q=2; p=2; newmv2fr;
>> iG=finv(w,mf); mimonyq(w,iG,2)
```

从图中可以看出，原系统并非对角占优的，这样可以首先引入一个矩阵 $\widehat{K}_{p_1}=G(0)$ 来对之进行补偿，这样可以得出如图 6-10 所示的带有列 Gershgorin 带的逆 Nyquist 曲线，可以看出该曲线的 $\hat{g}_{22}(s)$ 并不理想，可以通过选择 $\widehat{K}_{p_2}=[1,0;0.5,1]$ 对之进一步补偿，这样补偿后的 $\hat{g}_{22}(s)$ 逆 Nyquist 曲线如图 6-11 所示，可见补偿后的系统为对角占优。

```
>> Kp1=[0.1134,0.924; 0.3378,-0.318]; iG1=[];
```

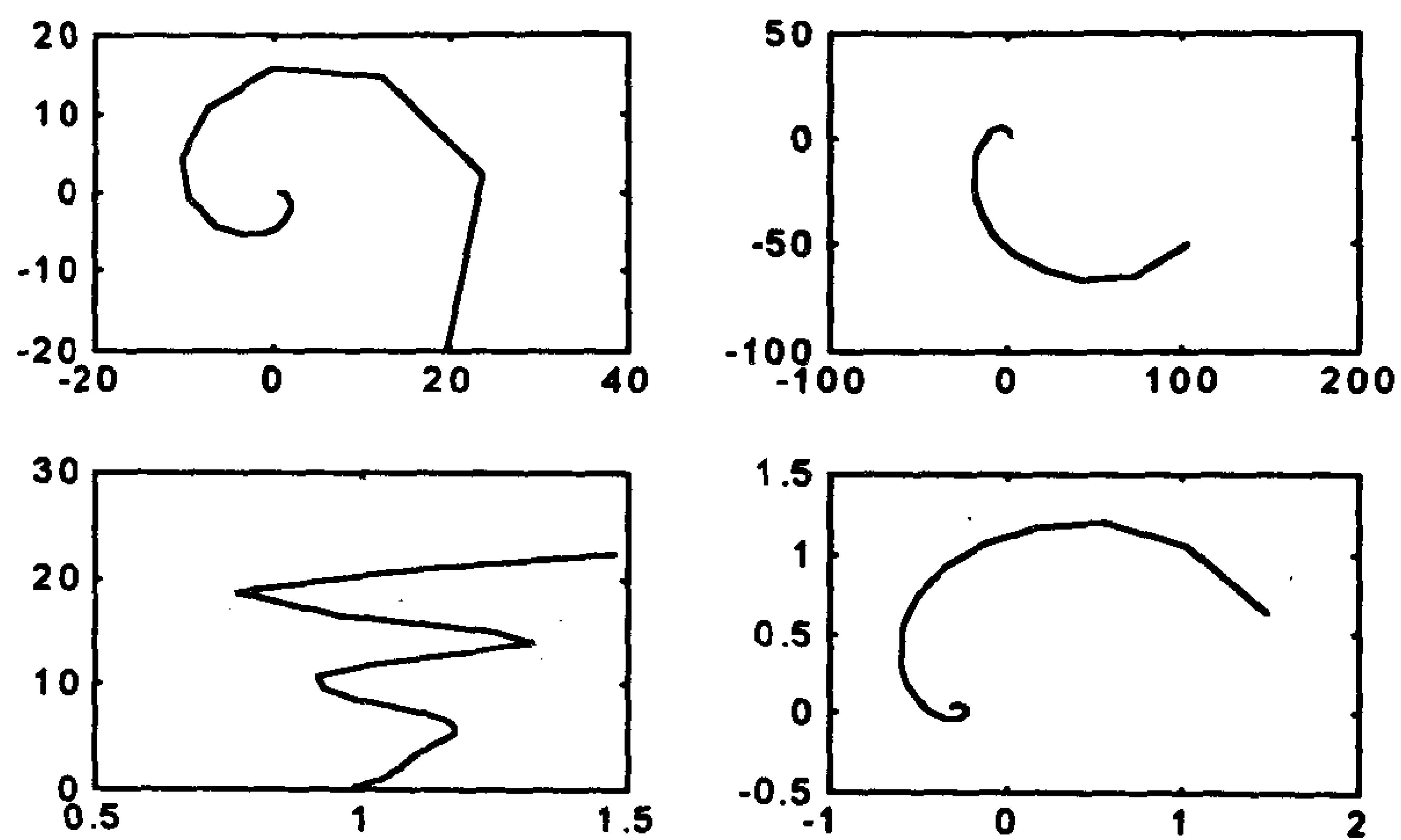


图6-9 原系统逆 Nyquist 图

```
>> for i=1:length(w), iG1=[iG1; Kp1*iG(2*i-1:2*i,:)]; end
>> mimonyq(w,iG1,2,2,1, [-15,20,0,35;-10 20,-20,5])
>> Kp2=[1,0; 0.5,1]; iG2=[];
>> for i=1:length(w), iG2=[iG2; Kp2*iG1(2*i-1:2*i,:)]; end
>> clg; plotnyq(fget(w,iG2,[2,2])), axis([-10,15,-30,5]); hold on;
>> C=frgersh(w,iG2,2); plotnyq(C,'-'), grid; hold off
```

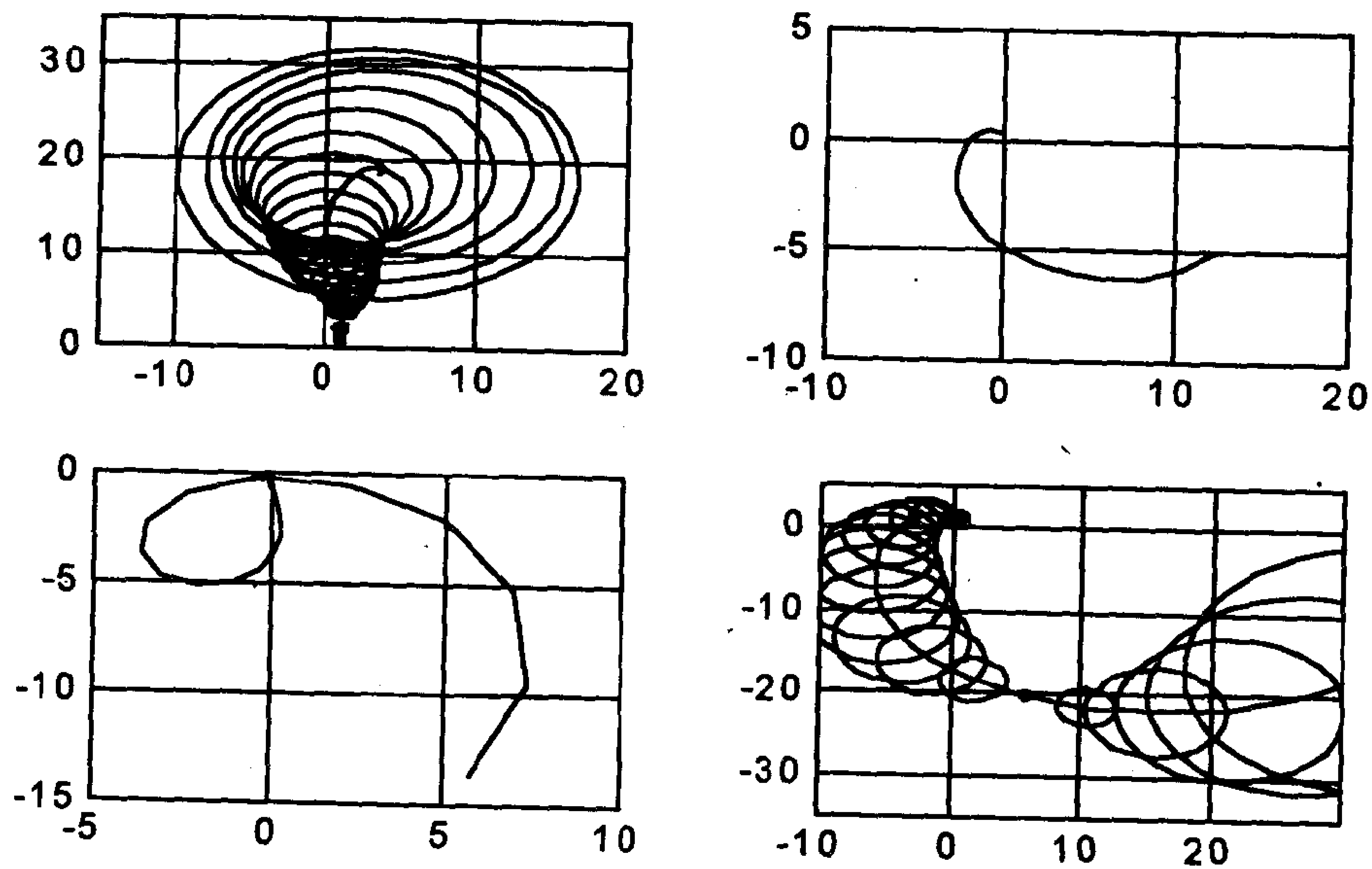


图6-10 $\hat{K}_{p_1}$ 补偿后的系统逆 Nyquist 图

对得出的对角占优系统这时可以利用单变量系统的设计方法对两个回路进行单独设计，例如引入下面的动态补偿矩阵

$$K_d(s)=\begin{bmatrix}1 & 0\\ 0 & (0.3s+1)/(0.05s+1)\end{bmatrix}，\ 所以有\ \widehat{K}_d(s)=\begin{bmatrix}1 & 0\\ 0 & (0.05s+1)/(0.3s+1)\end{bmatrix}$$

这时 $\widehat{Q}(s)=\widehat{K}_d(s)\widehat{K}_{p_2}\widehat{K}_{p_1}\widehat{G}(s)$ 的逆 Nyquist 图如图 6-12 所示，这时可以发现补偿后的系统有较强的对角占优特性。

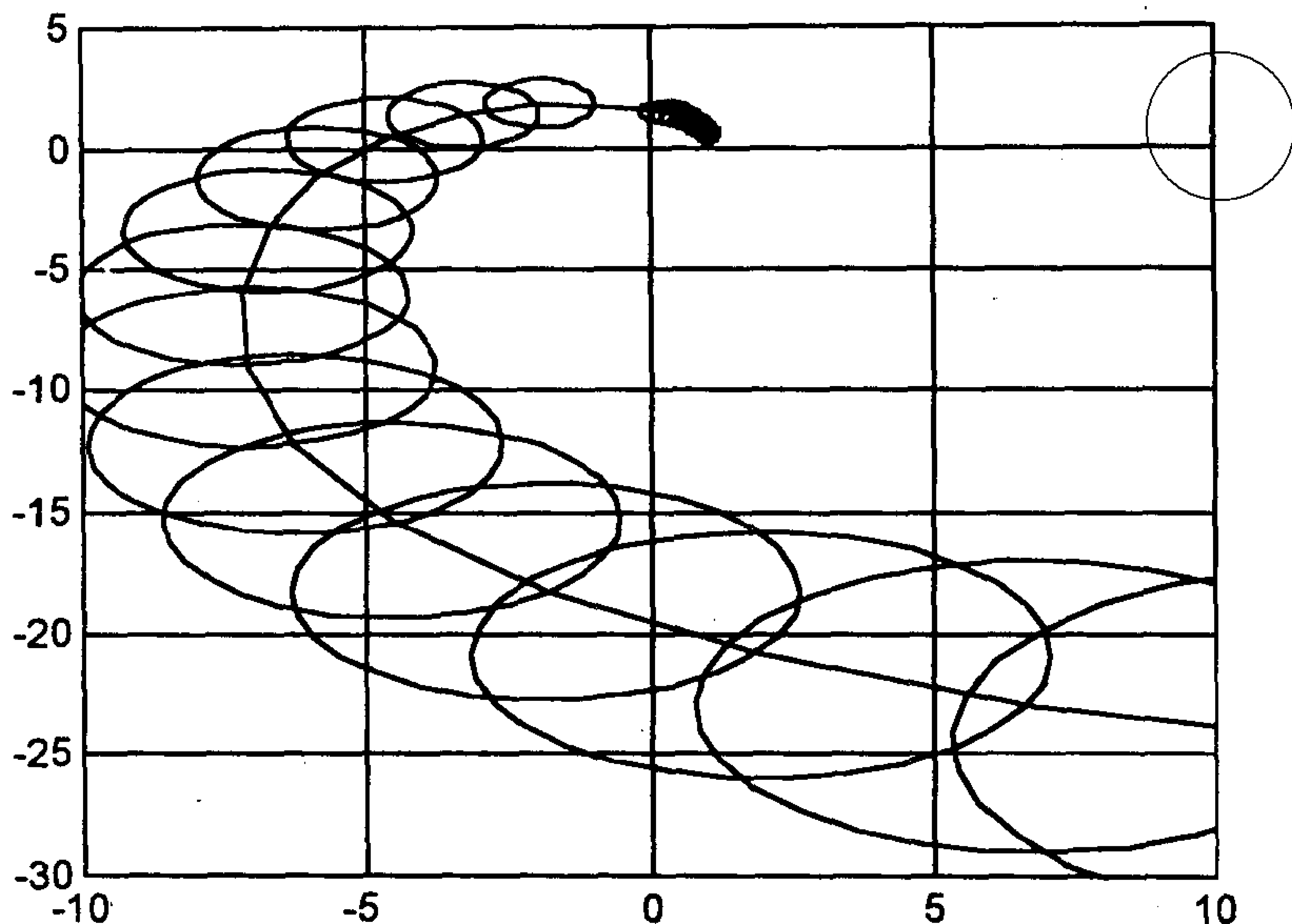


图6-11 $\widehat{K}_{p_2}\widehat{K}_{p_1}$ 补偿后的系统逆 Nyquist 图

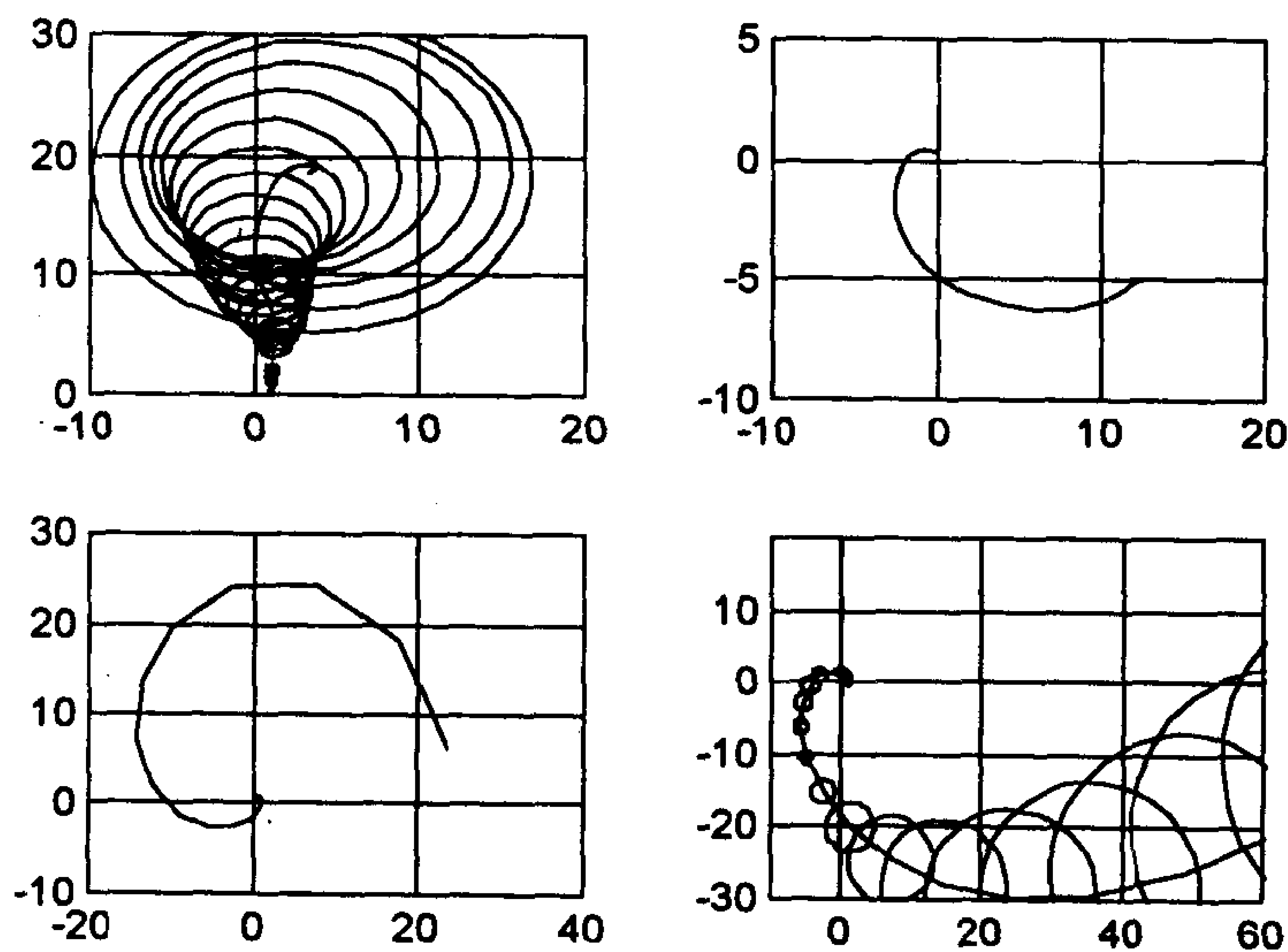


图6-12 $Q(s)$ 的逆 Nyquist 图

```
>> num_Gd=[0.3,1, 0,0; 0,0, 0.05,1]; den_Gd=[0.3,1];
>> fGd=mv2fr(num_Gd, den_Gd,w); iGd=finv(w,fGd); iG3=[];
```




```
>> for i=1:length(w)
     iG3=[iG3; iGd(2*i-1:2*i,:)*iG2(2*i-1:2*i,:)];
   end
>> mimonyq(w,iG3,2,2,1,[-10,20,0,30;-10,60,-30,20]);
```

若想求出闭环系统的阶跃响应，则应该首先按照图 6-13 的结构建立 SIMULINK 框图， 其中

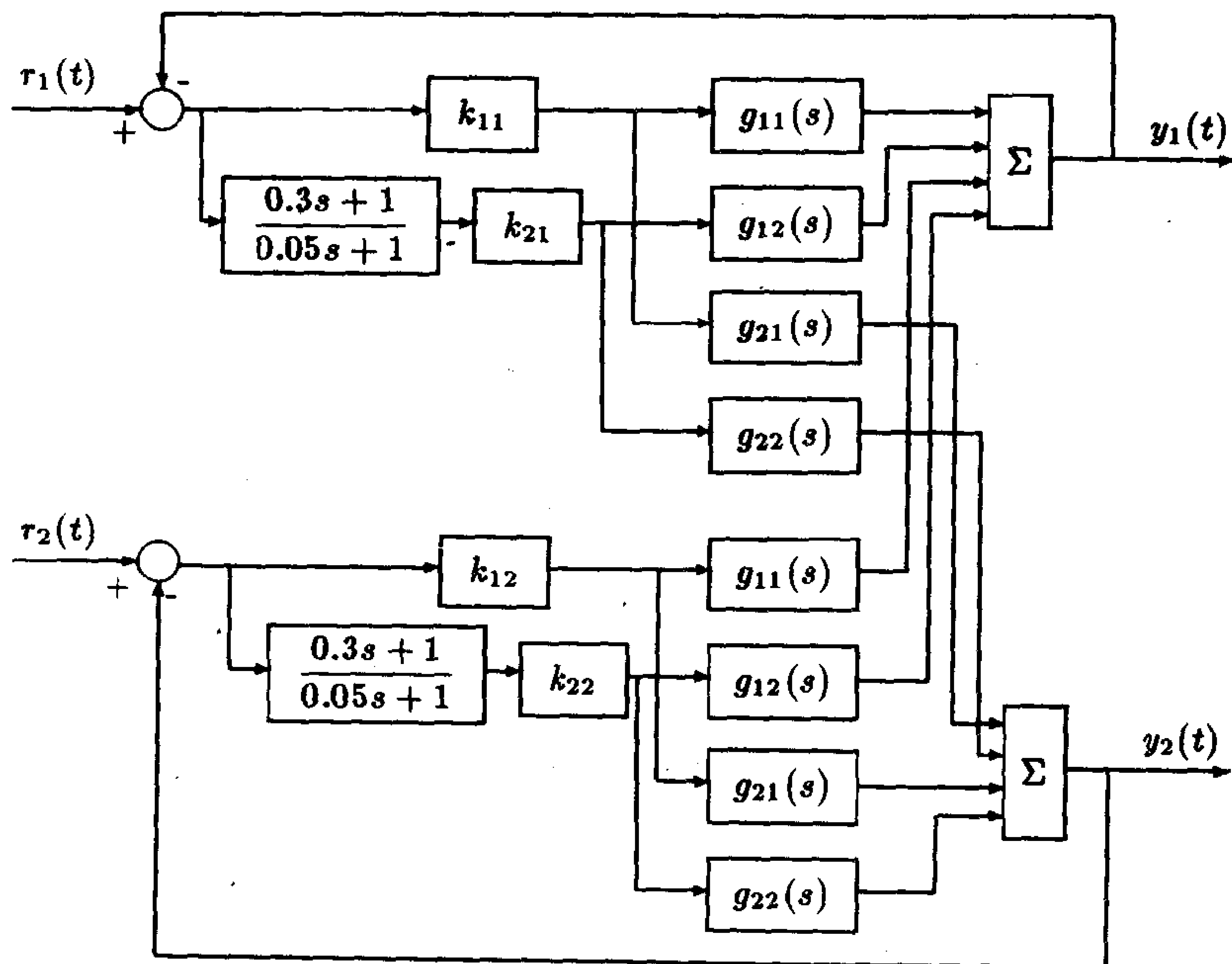


图6-13 多变量系统的仿真框图

对象模型 $g_{11}(s) = 0.1134\mathrm{e}^{-0.72s}/(1.78s^2 + 4.48s + 1)$, $g_{12}(s)$, $g_{21}(s)$ 和 $g_{22}(s)$ 亦有相应的定义，而 k 为 $K_{p_1}K_{p_2}$ 矩阵的参数。若想进行仿真，这样就可以由 SIMULINK 构造出如图 6-14 所示的仿真模型了，这时 k 参数应该设置为 `k=inv(Kp1)*inv(Kp2)` 即可。依据此模型就可以得出系统的阶跃响应曲线，如图 6-15 所示，若想由 SIMULINK 得出仿真结果，则首先应将 r1 设置为单位阶跃，将 r2 赋 0，这样可以得出输出信号 y1，y2 和时间信号 t，用 `t1=t; y11=y1; y12=y2` 命令将它们暂存起来，并将 r1 置为 0，将 r2 置为单位阶跃，这样就又可以得出一组数据，然后给出 `plot(t1, y11, t1, y12, t, y1, t, y2)` 命令就可以得出阶跃响应曲线。

在多变量系统设计时为了进一步改进对角占优性，还可以分别在各个反馈回路设置一个 f_i 的反馈补偿，这时可以证明，对应的特征值带 (即前面的 Gershgorin 带) 的半径为

$$r_i(s) = \phi_i(s)d_i(s), \quad 其中 \quad \phi_i(s) = \max_{j\neq i} \frac{d_j(s)}{|f_j + \hat{q}_{jj}(s)|} \tag{6.2.18}$$

式中 $d_i(s)$ 为原来 Gershgorin 圆的半径，这个带称为 Ostrowski 带。由得出的第 i 个 $\hat{g}_{ii}(s)$ 的 Gershgorin 带与负虚轴的交点 γ_i 可以确定出使系统稳定的负反馈系数 f_i 来，其实对每个回路只要选择 $f_i > \gamma_i$，则系统将是稳定的。可以证明，若原系统是对角占优的，则

引入使系统稳定的反馈 f_i 之后，得出的 Ostrowski 带的半径小于 Gershgorin 带，所以整个带会变得更窄，该式中 $\phi_i(s)$ 称为收缩比例 (shrinking ratio)。

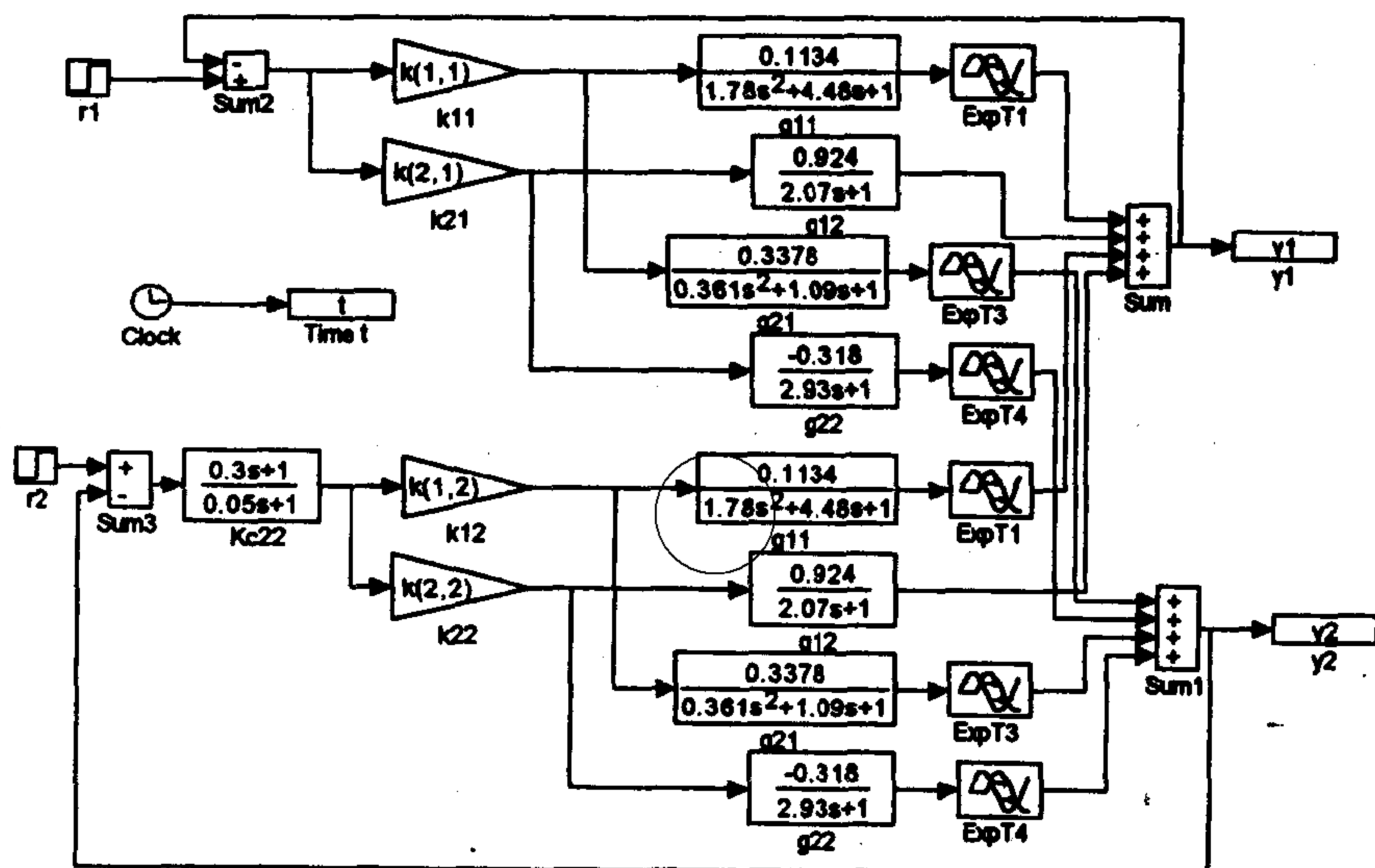


图6-14　多变量系统仿真的 SIMULINK 模型

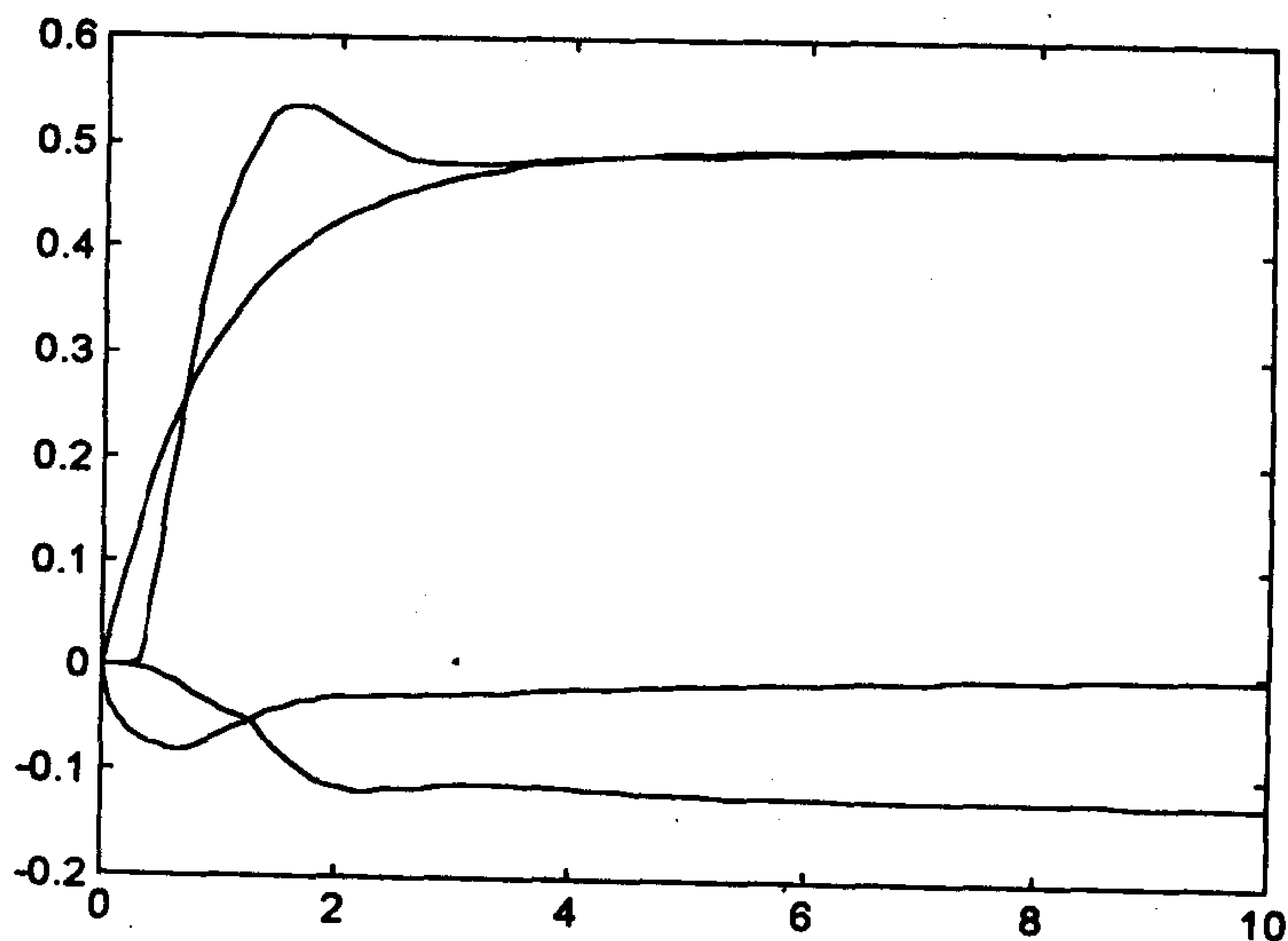


图6-15　闭环系统的阶跃响应曲线

MFD 工具箱提供了求取 Ostrowski 带的函数 `fcost()` 和 `frost()`，分别用来绘制列和行 Ostrowski 带，例如 `fcost()` 函数的调用格式为

```
points=fcost(w, iG, iu, k)
```

其中 w 和 iG 分别为频率向量和频率响应矩阵，iu 为输入代号，k 为该输入下的反馈系数，由此函数可以得出各个 Ostrowski 圆的坐标点 points，可以用类似于 fcgersh() 函数得出的数据那样绘制出 Ostrowski 带。

在前面的例子中，可以给出两个反馈系数 $f_1 = 5.0$, $f_2 = 3.5$, 这时设计后系统的 Ostrowski 带可以由图 6-16 表示，该图形是由下面的 MATLAB 命令绘制出来的

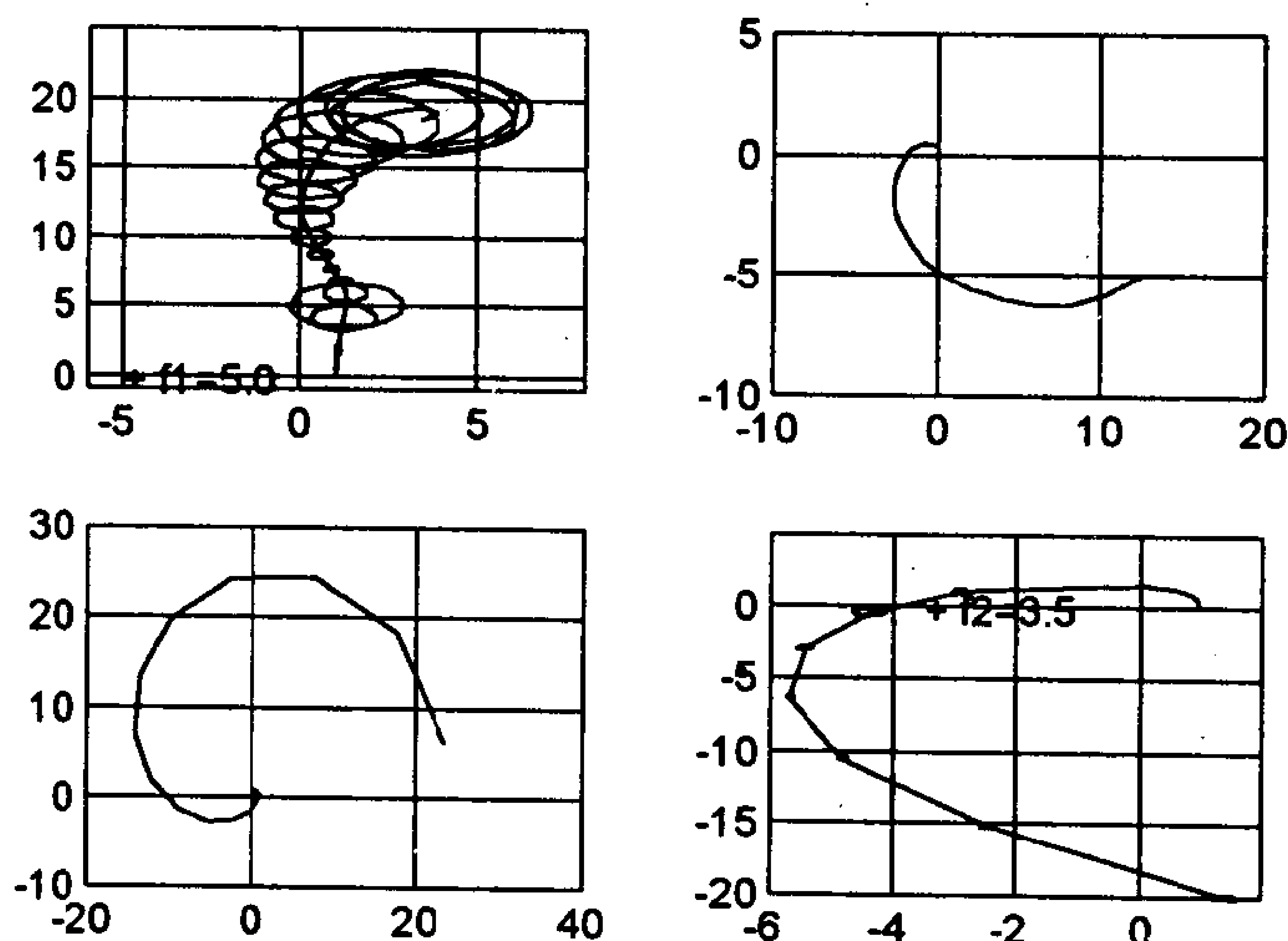


图6-16　多变量系统的 Ostrowski 带

```
>> ff=[5,3.5]; pt1=frost(w,iG3,1,ff); pt2=frost(w,iG3,2,ff);
>> subplot(221); plotnyq(fget(w,iG3,[1,1])); axis([-6,8,-1,25]);
>> hold on; plot(pt1,'-'); text(-5,0,'+ f1=5.0'); grid; hold off
>> subplot(222); plotnyq(fget(w,iG3,[1,2])); grid
>> subplot(223); plotnyq(fget(w,iG3,[2,1])); grid
>> subplot(224); plotnyq(fget(w,iG3,[2,2])); axis([-6,2,-20,5]);hold on;
>> plot(pt2,'-'); text(-3.5,0,'+ f2=3.5'); grid; hold off
```

这样整个系统的闭环阶跃曲线如图 6-17 所示，可见系统的阶跃响应被明显地改进了。

其实对多变量系统进行对角优势化的方法并不是唯一的，前面介绍的方法主要是试凑的方法，利用上节中介绍的伪对角化方法，选择频率 $\omega = 1$ 对 $\widehat{K}_{p_1}\widehat{G}(s)$ 也可以由下面的语句进行伪对角化

```
>> iG4=iG1(27:28,:); K=pseudiag(iG4); iG5=[];
>> for i=1:length(w), iG5=[iG5; K*iG1(2*i-1:2*i,:)]; end
>> mimonyq(w,iG5,2,2,1,[-2,5,-1,25; -50,10,-5,30])
```

其中 iG1 的第 27 和 28 行对应于 $\omega = 1$ 的矩阵，这样就可以得出如图 6-18 所示的逆 Nyquist 曲线。可见此系统也是对角占优的，对此系统也可以进一步校正得出满意的阶跃响应曲线。

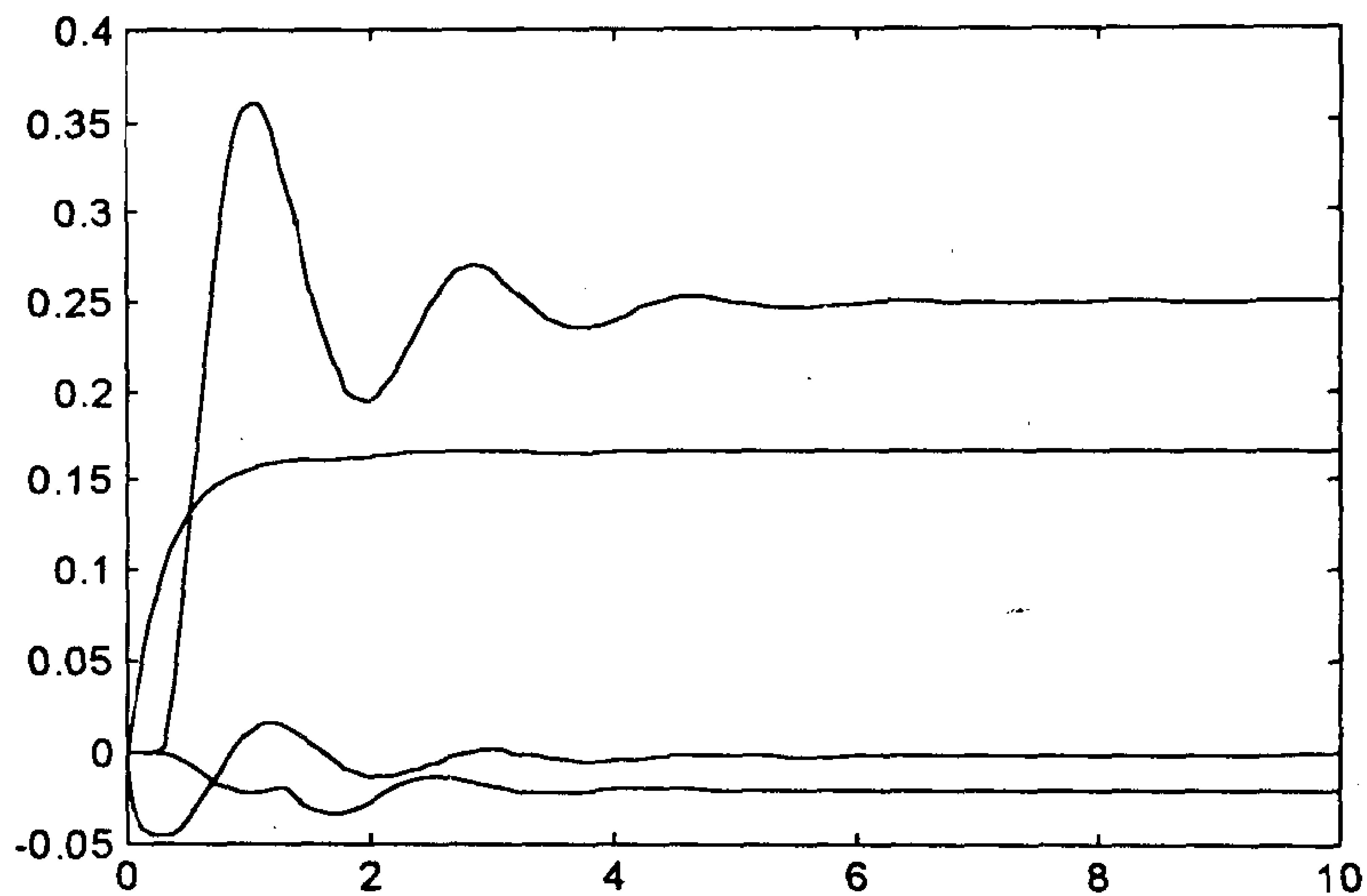


图 6-17 最终闭环系统的阶跃响应

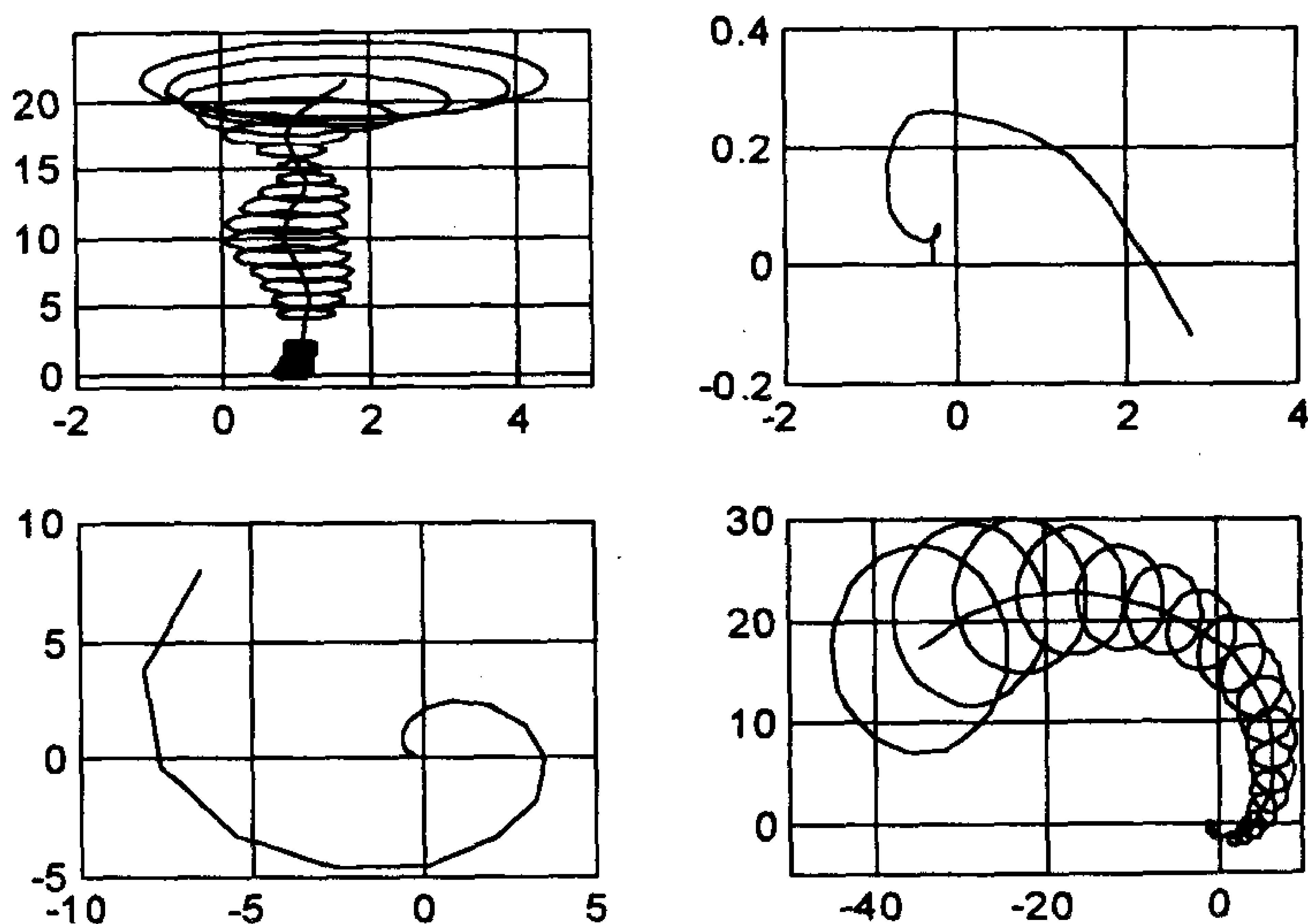


图 6-18 伪对角化后的逆 Nyquist 图形

6.3 多变量系统的其它设计方法

多变量系统的频域设计中除了前面介绍的逆 Nyquist 阵列方法之外，比较流行的还有特征轨迹法 (characteristic locus method) [22]，反标架坐标法 (reversed-frame normalisation, 简称 RFN) [18]，序贯回路闭合方法 (sequential loop closing) [25]，此外还有参数最优化方法 (parameters optimisation method)[10] 及不等式方法 (method of inequalities) [34] 等，在这里将介绍最常用的特征轨迹法和参数最优化方法，其它方法可以查阅有关文献 (如文献 [24])。

6.3.1 多变量系统的特征轨迹方法

假设传递函数矩阵 $G(s)$ 为一个 $m \times m$ 方阵，并假定在 s 下的 $G(s)$ 矩阵特征值为 $\lambda_i(s)$ $(i=1,\cdots,m)$，则 $\lambda_i(s)$ 随 s 变化的的值可以构成 m 个轨迹，它们称为 $G(s)$ 的特征轨迹 (characteristic locus)。显而易见，假设在负反馈结构下存在一个补偿矩阵 $K(s)=kI$，可以证明，若 $\lambda_i(s)$ 是 $G(s)$ 矩阵的一个特征值，则 $k\lambda_i$ 是 $kG(s)$ 矩阵的特征值，且 $1+k\lambda_i(s)$ 为 $I+kG(s)$ 矩阵的特征值。

由特征轨迹可以很容易地判断闭环系统的稳定性，这可以根据下面的定理[21]来进行。

若 $G(s)$ 有 P_o 个不稳定 Simth-McMillan 极点，返回比为 $-kG(s)$ 的闭环系统稳定的充要条件为 $kG(s)$ 的特征轨迹逆时针地围绕 (-1,0) 点 P_o 次。

例 6.6 假设系统的传递函数矩阵为

$$G(s)=\frac{1}{1.25(s+1)(s+2)}\begin{bmatrix} s-1 & 2 \\ -6 & s-2 \end{bmatrix}$$

记 $\Delta(s)=1.25(s+1)(s+2)$，则由 $\det\{\lambda I+G(s)\}=0$ 可以得出下面关于 λ 的一元二次方程

$$\lambda^2-\frac{2s-3}{\Delta(s)}\lambda+\frac{(s+1)(s+2)}{\Delta^2(s)}=0$$

求解此方程则可以直接得出

$$\lambda_{1,2}=\frac{1}{2\Delta(s)}\left(2s-3\pm\sqrt{1-24s}\right)$$

故用下面命令容易地绘制出系统的特征轨迹，如图 6-19 所示。

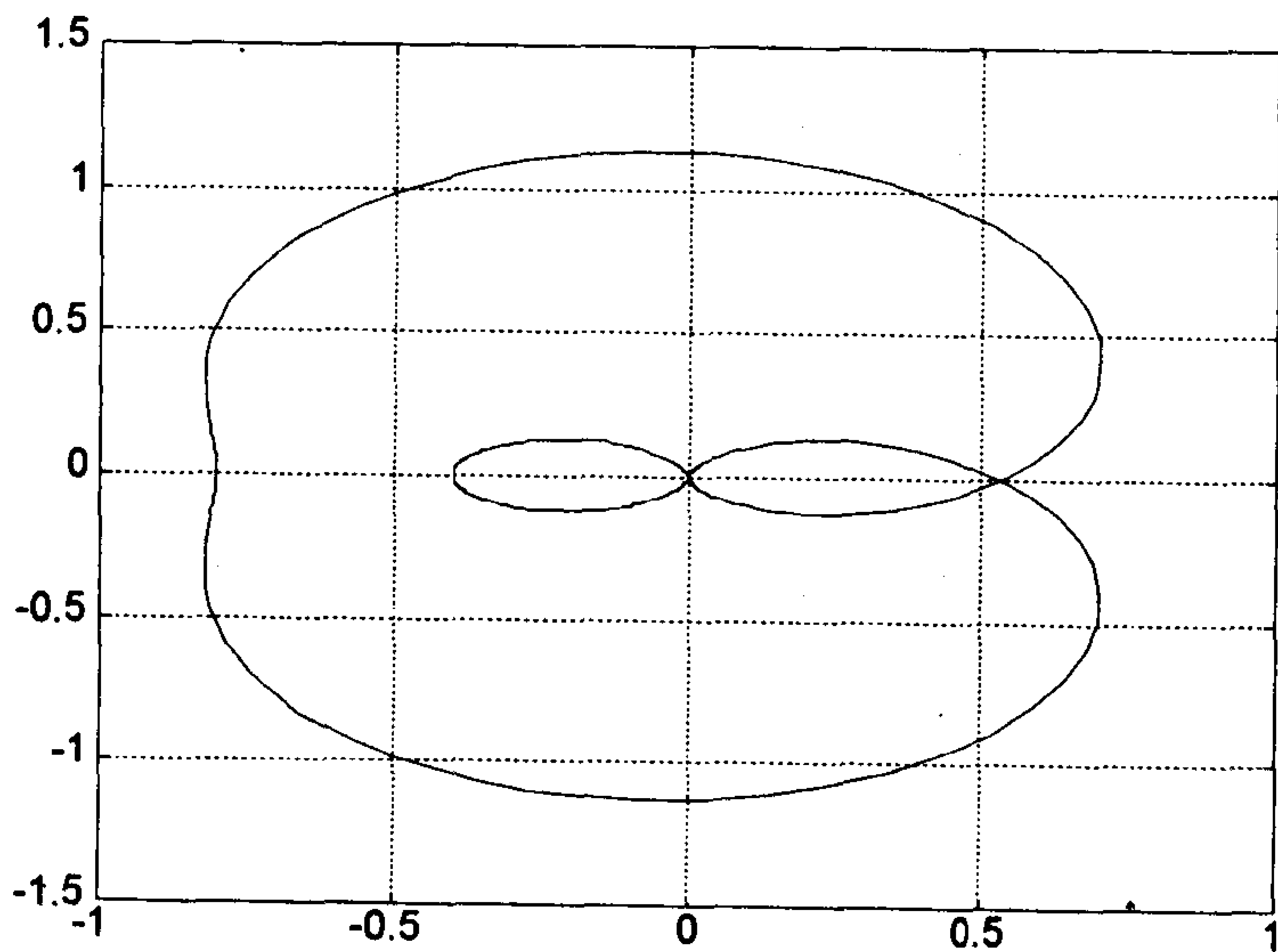


图6-19 给定系统的特征轨迹表示

```
s=sqrt(-1)*logspace(-2,2,100); s=[-s(length(s):-1:1) s]; dd=2*1.25*(s+1).*(s+2);
xy1=(2*s-3+sqrt(1-24*s))./dd; xy2=(2*s-3-sqrt(1-24*s))./dd;
plot(real(xy1),imag(xy1),real(xy2),imag(xy2)), grid
```

由原系统模型已知，开环极点全部是稳定的，所以从得出的曲线可以看出，若 $-\infty < -1/k < -0.8$, $-0.53 < -1/k < \infty$ 或 $-0.4 < -1/k < 0$, 则特征轨迹并不包含 $(-1,0)$ 点 [1)]，故闭环系统将是稳定的，若 $-0.8 < -1/k < -0.4$, 则特征轨迹包含一次 $(-1,0)$ 点，故系统不稳定，而 $0 < -1/k < 0.53$ 时特征轨迹包含两次 $(-1,0)$ 点，故闭环系统仍然是不稳定的。

MATLAB 的多变量频域设计工具箱提供了求取频域响应矩阵特征值的函数 feig()，并可以该函数得出的结果进行进一步处理，得出连续的特征轨迹图形，该功能是由 csort() 函数而完成的。这里 feig() 和 csort() 函数的调用格式分别为

```
V0=feig(w,mf)    或    V1=csort(V0)
```

其中 w 为频率向量，其实也可以和前面一样扩展到含有负“频率”值的情况，mf 是按照 MFD 规则得出的频率响应数据矩阵，由这两个矩阵则可以返回各个频率下特征值向量构成的矩阵 V0，然后由矩阵 V0 调用 csort() 函数就可以依据某种原则由离散点数据构造出连续的数据向量，以使得可以直接调用 plot() 函数来绘图。

下面我们将对这个例子用 MFD 工具箱进行重新绘制，则结果和图 6-19 所示的完全一致。

```
den=1.25*conv([1,1],[1,2]); num=[0 1 -1 0 1 0; 0 0 -6 0 1 -2];
w=logspace(-2,2,100); w=[-w(length(w):-1:1) w]; Gf=mv2fr(num,den,w);
V=feig(w,Gf);  V1=csort(V); plot(V1); grid
```

假设原系统的传递函数矩阵是 $m \times m$ 方阵，即系统的输入与输出个数相同 (均为 m)，这样往往可以将系统分解为 $G(s) = W(s)\Lambda(s)W^{-1}(s)$, 其中 $W(s)$ 矩阵可以为 $G(s)$ 的特征向量构成的矩阵，而 $\Lambda(s) = \mathrm{diag}\{\lambda_1, \cdots, \lambda_m(s)\}$ 为 $G(s)$ 的特征值构成的对角矩阵。若控制器选择为 $K(s) = W^{-1}(s)M(s)W(s)$, 其中 $M(s) = \mathrm{diag}\{\mu_1(s), \cdots, \mu_m(s)\}$, 则返回比矩阵可以写成

$$-G(s)K(s) = -W\Lambda(s)M(s)W^{-1}(s) = -W(s)N(s)W^{-1}(s) \tag{6.3.1}$$

其中 $N(s) = \mathrm{diag}\{v_1(s), \cdots, v_m(s)\}$, 且 $v_i(s) = \lambda_i(s)\mu_i(s)$ 。这样构成的补偿器 $K(s)$ 是可交换的，又称为可交换补偿器 (commutative compensator) 因为返回比矩阵满足 $-G(s)K(s) = -K(s)G(s)$, 这样问题似乎是很显然的，可以依照前面的方法来直接构造补偿器 $K(s)$, 然而事实并非如此简单，因为 $W(s)$ 或 $W^{-1}(s)$ 并不能保证为正则函数，所以一般不能直接实现 $K(s)$ 。这时若能选择可实现函数 $A(s) \simeq W(s)$, 且 $B(s) \simeq W^{-1}(s)$, 则可以构造出具有相似特性的近似可交换补偿器 (approximate commutative compensator), 在一般应用中往往选择 $A(s)$ 和 $B(s)$ 为实数矩阵，记 $A = [a_1, a_2, \cdots, a_m]$, 这里 a_i 为列向量，文献 [20] 提出了获取实系数矩阵 A 的 ALIGN 算法，其目的是找出一个可以匹配于复数矩阵的实数矩阵，假设在某一个 $s = s_0$ 下 $W(s_0)$ 可以写成 $W(s_0) = [w_1, w_2, \cdots, w_m]$, 则若存在复数标量值 z_i, 使得 $a_i = w_i z_i$, 则可以在 s_0 处构造出可交换补偿器，这时 $B = A^{-1}$ 。若定义 $V(s) = W^{-1}(s)$, 且记 $V^T(s_0) = [v_1, v_2, \cdots, v_m]$, 则 $i \neq j$ 时 $v_j^H a_i = 0$,

1)尤其注意后一种情况，这时看似包含该点，实际上包含该点的总数为 0 。

这时可以根据下式来求取 A 矩阵

$$a_i = \arg\max_{a_i} \frac{|v_i^H a_i|^2}{\sum_{j\neq i} |v_j^H a_i|^2} \tag{6.3.2}$$

改变 a_i 值，使得指标式取最大值，并将这时的向量 a_i 提取出来，然后根据这样的 m 个向量构造出 A 矩阵。其实实现 ALIGN 算法需要求解矩阵广义特征值问题，多变量频域设计工具箱中提供了 ALIGN 算法的函数 align()，其基本部分如下：

```
function kpc=align(f)
ff=f'*f;  rff=real(ff);
x=f*(rff\conj(ff'))*(rff\conj(f'));
p=angle(diag(x))/2;
kpc=rff\(real(f'*diag(exp(sqrt(-1)*p))));
```

特征轨迹设计方法分 4 个步骤来实现：

- **高频设计**: 首先选择一个频率点 ω_b，并设计一个高频实矩阵补偿器 $K_h \simeq -G^{-1}(j\omega_b)$，显然这需要使用 ALIGN 算法来实现。其中频率 ω_b 往往这样选定，绘制原系统的特征轨迹，并找出它与 $M = 1/\sqrt{2}$ 的等 M 圆交点处的频率值赋给 ω_b。
- **中频设计**: 在某个 $\omega_m < \omega_b$ 频率下对初步补偿的对象 $G(s)K_h$ 设计中频补偿器 $K_m(s)$。
- **低频设计**: 一般情况下低频部分 (选择 $\omega_l < \omega_m$) 应包含一个积分器矩阵 I_m/s，这时在 ω_l 处对 $G(s)K_hK_m(s)$ 进行设计，最终得出低频补偿器 $K_l(s)$。
- **补偿器的实现**: 设计出各个频率段的补偿器之后，则总的补偿器可以写成

$$K(s) = K_hK_m(s)K_l(s) \tag{6.3.3}$$

例 6.7 考虑单变量频域设计工具箱中给出的例子 [9]，假设原系统模型由下面的状态方程给出

```
>> a=[0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000;
      0.00000 -.11323 -.98109 -11.847 -11.847 -63.080 -34.339 -34.339 -27.645 0.00000;
      324.121 -1.1755 -29.101 0.12722 2.83448 -967.73 -678.14 -678.14 0.00000 -129.29;
      -127.30 0.46176 11.4294 -1.0379 13.1237 380.079 266.341 266.341 0.00000 1054.85;
      -186.05 0.67475 16.7045 0.86092 -17.068 555.502 389.268 389.268 0.00000 -874.92;
      341.917 1.09173 1052.75 756.465 756.465 -29.774 0.16507 3.27626 0.00000 0.00000;
      -30.748 -.09817 -94.674 -68.029 -68.029 2.67753 -2.6558 4.88497 0.00000 0.00000;
      -302.36 -.96543 -930.96 -668.95 -668.95 26.3292 2.42028 -9.5603 0.00000 0.00000;
      0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 -1.6667 0.00000;
      0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 -10.000];
>> b=[0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.66667 0.00000;
      0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 10.0000]';
>> c=[1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.0000;
      -.49134 0.00000 -.63203 0.00000 0.00000 -.20743 0.00000 0.00000 0.00000 0.0000];
>> d=[0 0; 0 0];
```

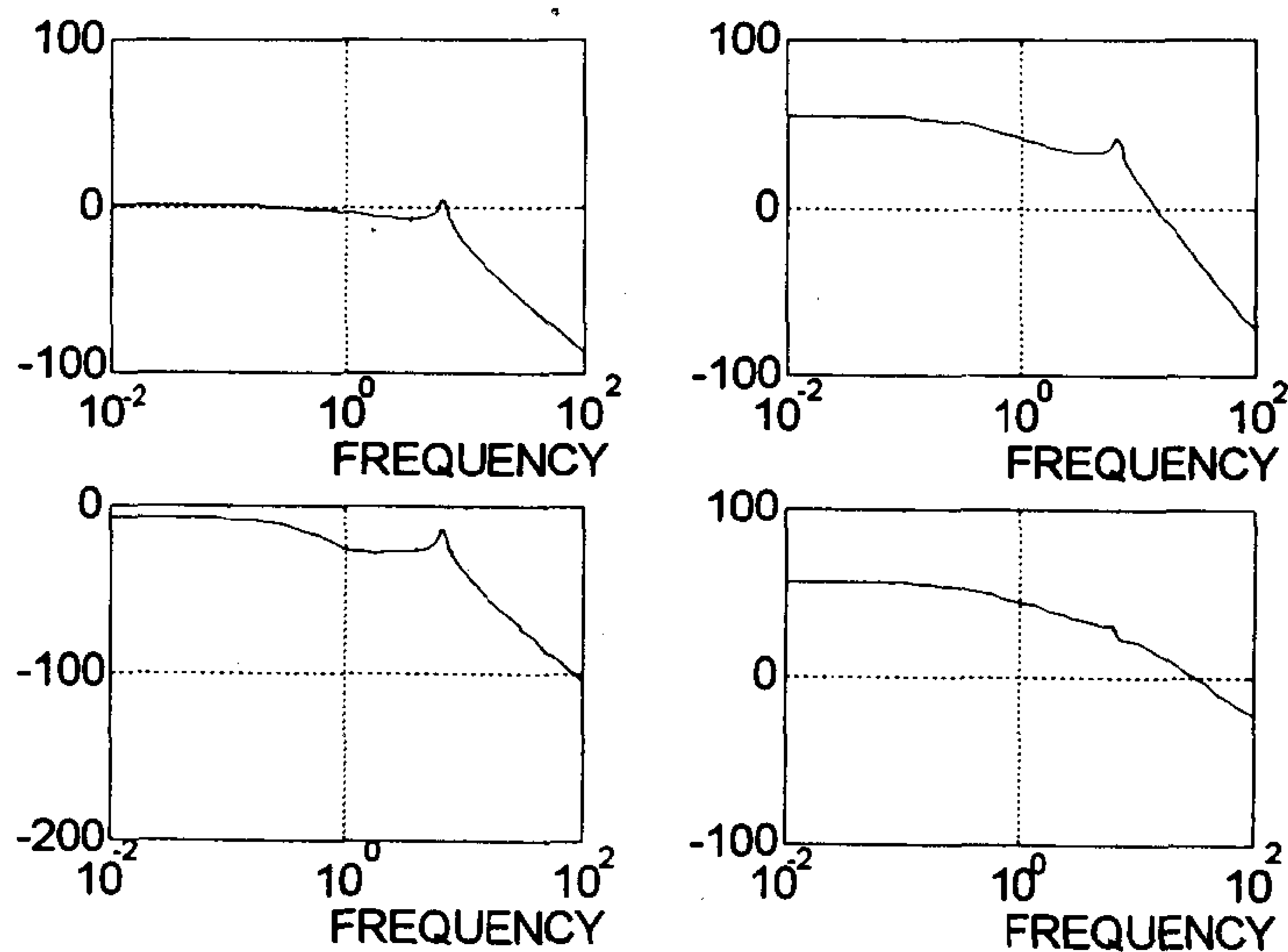


图6-20 原系统的 Bode 曲线

下面将考虑用特征轨迹方法来设计出控制器来，首先构造频率向量 w, 并将频率点 $\omega = 6.34$ 和 $\omega = 30$ 插入频率向量，因为在后面的设计中还需使用其值。构造频率向量之后，则可以得出系统的频率响应数据，并绘制出 Bode 曲线，如图 6-20 所示。

```
>>  w=logspace(-2,2,40); w2=[110:172:990]/100; w3=[55:3:77]/10; w4=6.34; w5=30;
>>  w=sort([w w2 w3 w4 w5]); g=mv2fr(a,b,c,d,w);
>>  subplot(221), mvdb(w, g, [1,1]), subplot(222), mvdb(w, g, [1,2]),
>>  subplot(223), mvdb(w, g, [2,1]), subplot(224), mvdb(w, g, [2,2]),
```

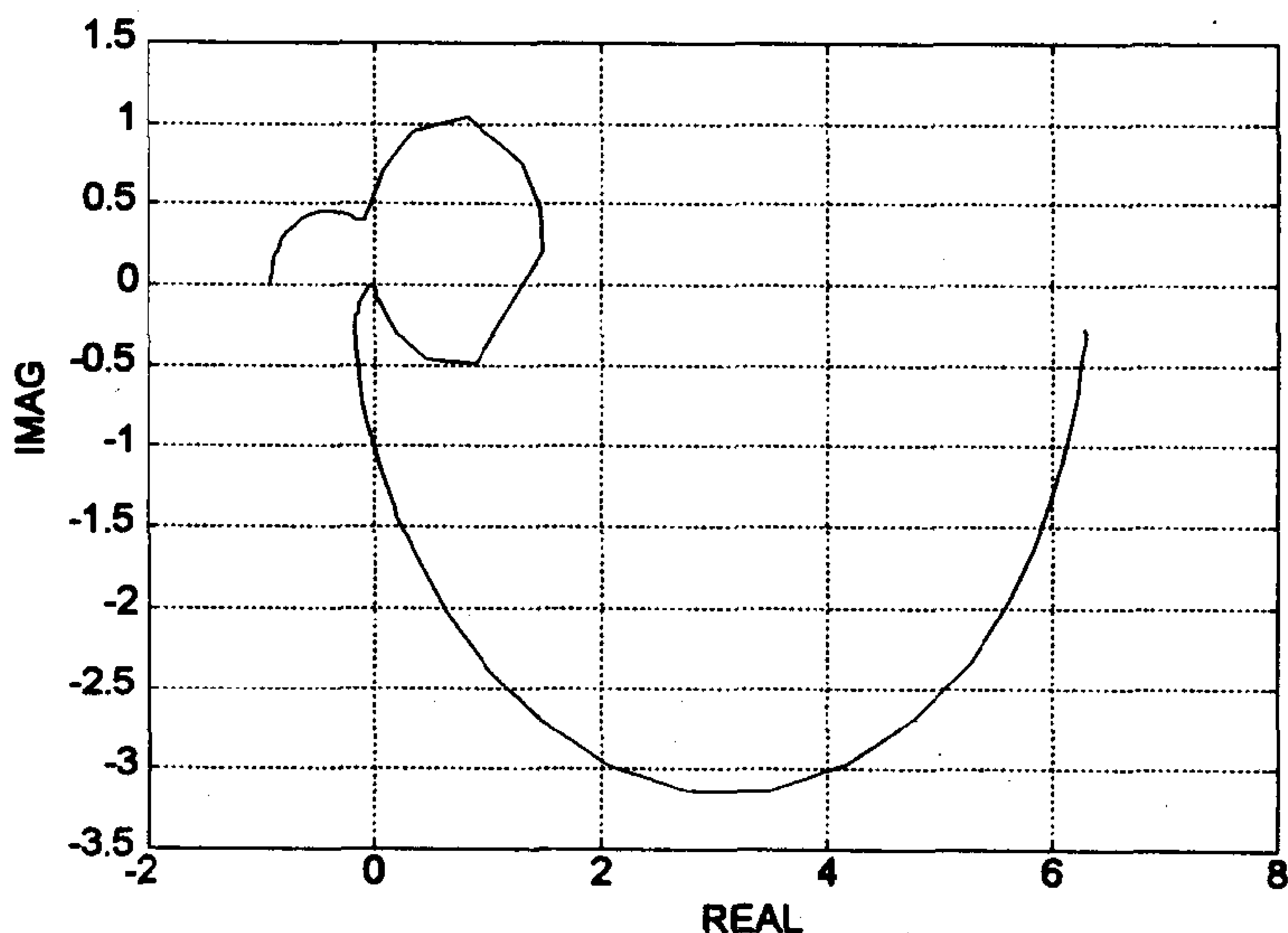


图6-21 k_1 校正后系统的特征轨迹表示

由图中可以看出，第 2 个输入的增益远大于第 1 输入的增益，所以可以如下选择 k1, 使得其值减小 100 倍，并绘制出初步校正后的系统特征轨迹曲线，如图 6-21 所示。

```
>>  k1 = [1, 0; 0, 0.01]; gk = fmul(w, g, k1);
>>  ev = feig(w, gk); ev = csort(ev); plotnyq(ev)
```

选择 $\omega_b=30$, 求出高频处的实 ALIGN 矩阵 k_2 并对之进行简单的变换则可以得出高频校正后系统的特征轨迹曲线，如图 6-22 所示。

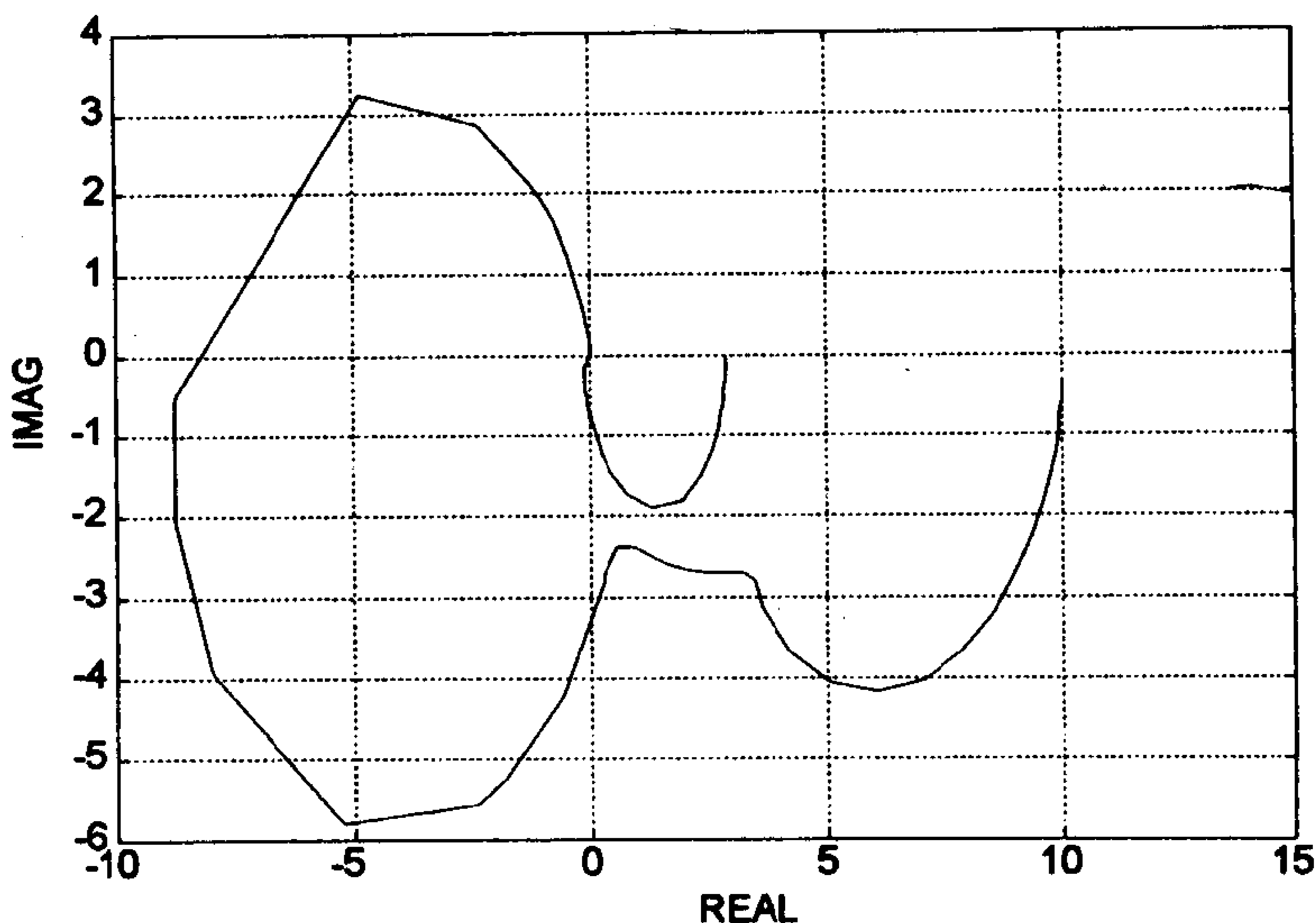


图 6-22　高频校正后系统的特征轨迹表示

```
>>  index=min(find(w>=30)); [gk30,w30]=fgetf(w,gk,index); k2 = align(gk30),
k2 =  549.4682   -1.3562
       -0.9005  -89.7454
>>  km = [-0.01,0; 0,-0.01]; k2 = k2*km
k2 =    -5.4947     0.0136
         0.0090     0.8975
>>  gk = fmul(w, gk, k2); ev = feig(w, gk); ev = csort(ev); plotnyq(ev)
```

还可以由 Malign=fmisalg(w, gk); semilogx(w, Malign) 命令检验在 $\omega_b>30$ 时的角度失配 (angle misalignment) 曲线，如图 6-23 所示。可见在 $\omega>10$ 时二者的角度匹配就已经很好了。

对中频部分可以引入一个近似可交换控制器，其中对第 1 输入和第 2 输入的补偿器分别为

$$k_{c1}(s)=\frac{(s^2+0.7s+40.37)(s+1)}{(s^2+45s+500)(0.1s+1)},\quad k_{c2}(s)=\frac{0.5s+1}{0.01s+1}$$

这时可以调用 facc() 函数来构造近似可交换控制器 ka, kb, kc, kd, 并得出校正后系统的特征轨迹曲线，如图 6-24 所示。

```
>>  kn1=[1 0.7 40.37]; kd1=[1 45 500]; kn1=conv(kn1,[1 2]); kd1=conv(kd1,[0.1 1]);
>>  kn2=[0.5 0.1]; kd2=[0.01 1]; kn=[kn1; zeros(1,2) kn2]; kd=[kd1; zeros(1,2) kd2];
>>  [ka,kb,kc,kd] = facc(w,gk,index,kn,kd); kf = mv2fr(ka,kb,kc,kd,w);
>>  gk = fmulf(w,gk,kf); ev = feig(w,gk); ev = csort(ev);  plotnyq(ev)
```

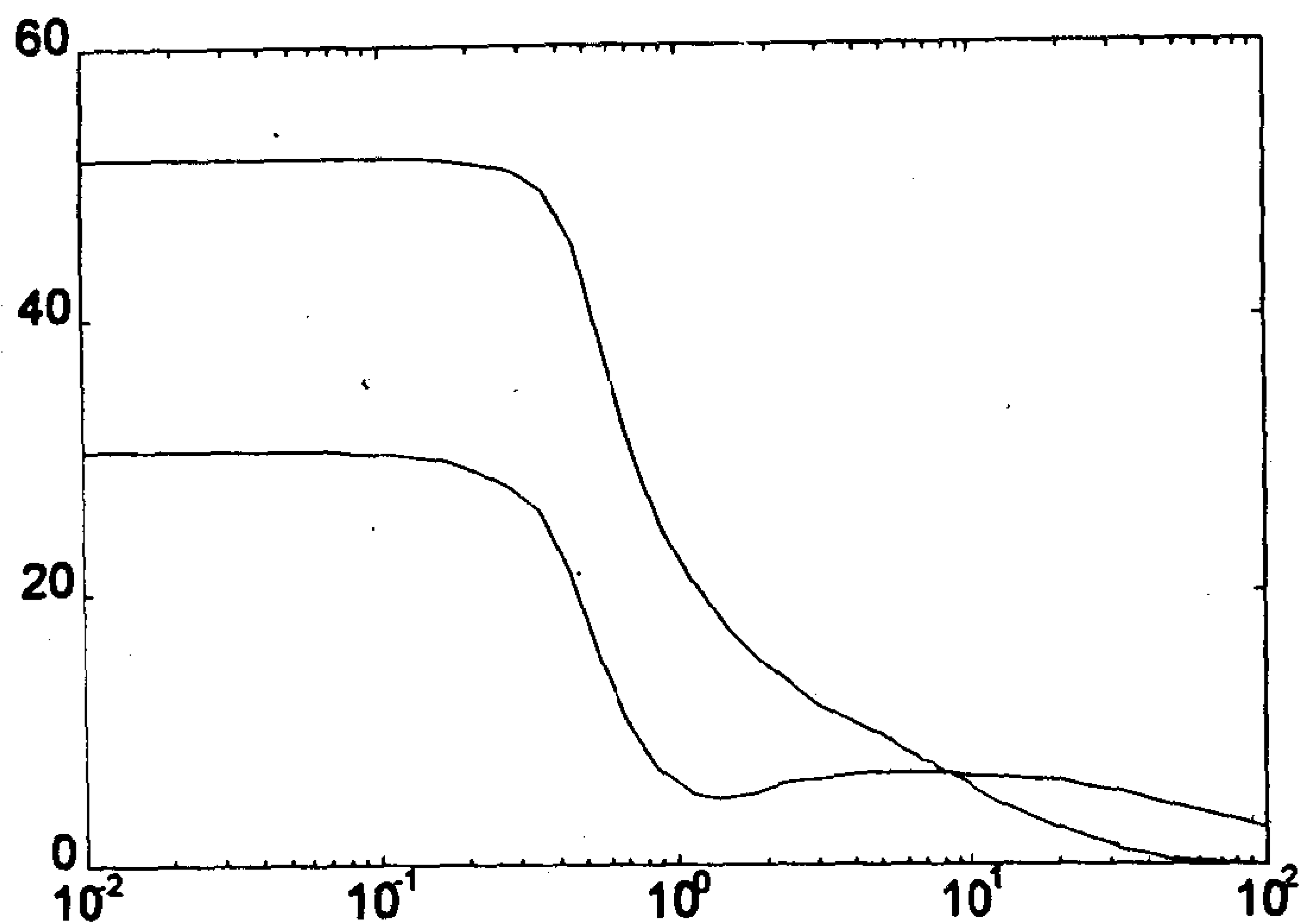


图6-23 频率响应角度失配显示

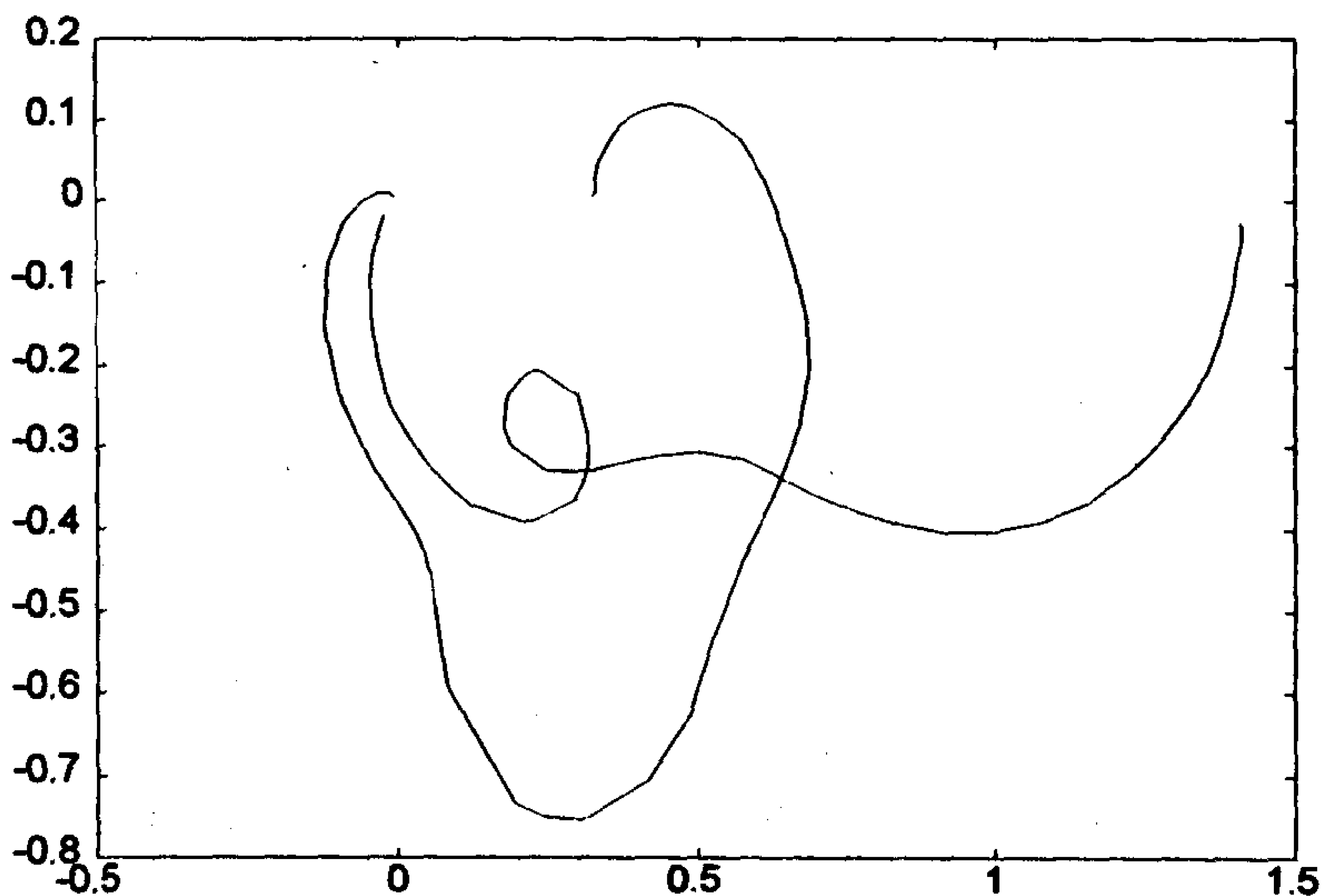


图6-24 引入中频近似可交换控制器后特征轨迹

由特征轨迹可见，低频时的增益是很低的，所以应该在不减小稳定裕度和已经得到的解耦特性的前提下增大其低频增益。这里可以采用传统的 PI 控制来构造低频近似可交换控制器，低频设计还是比较繁琐的，可以采用 MFD 工具箱中建议的方法如下设计控制器，

```
>>  gk1=fgetf(w, gk, 1); [vf, deigf]=eig(gk1); k6=align(inv(vf)); k8=align(vf);
>>  k7=abs(eig(gk1));  k7=diag(k7.\1); klow=k6*k7*k8; alpha=6.34/max(max(k7));
>>  klow = alpha * klow; [nklow, dklow]=pm2tf( [1 0], klow, eye(2));
>>  fklow = mv2fr(nklow, dklow, w); gk = fmulf(w,gk,fklow);
>>  k9=[3,0;  0,3]; gk=fmul(w,gk,k9); ev=csort(feig(w,gk));
>> plotnyq(ev), hold on, plotnyq(mcirc([1,2])); hold off; axis([-5, 5, -5, 5]);
```

构造出控制器后，则可以绘制出系统的特征轨迹曲线，如图 6-25 所示。

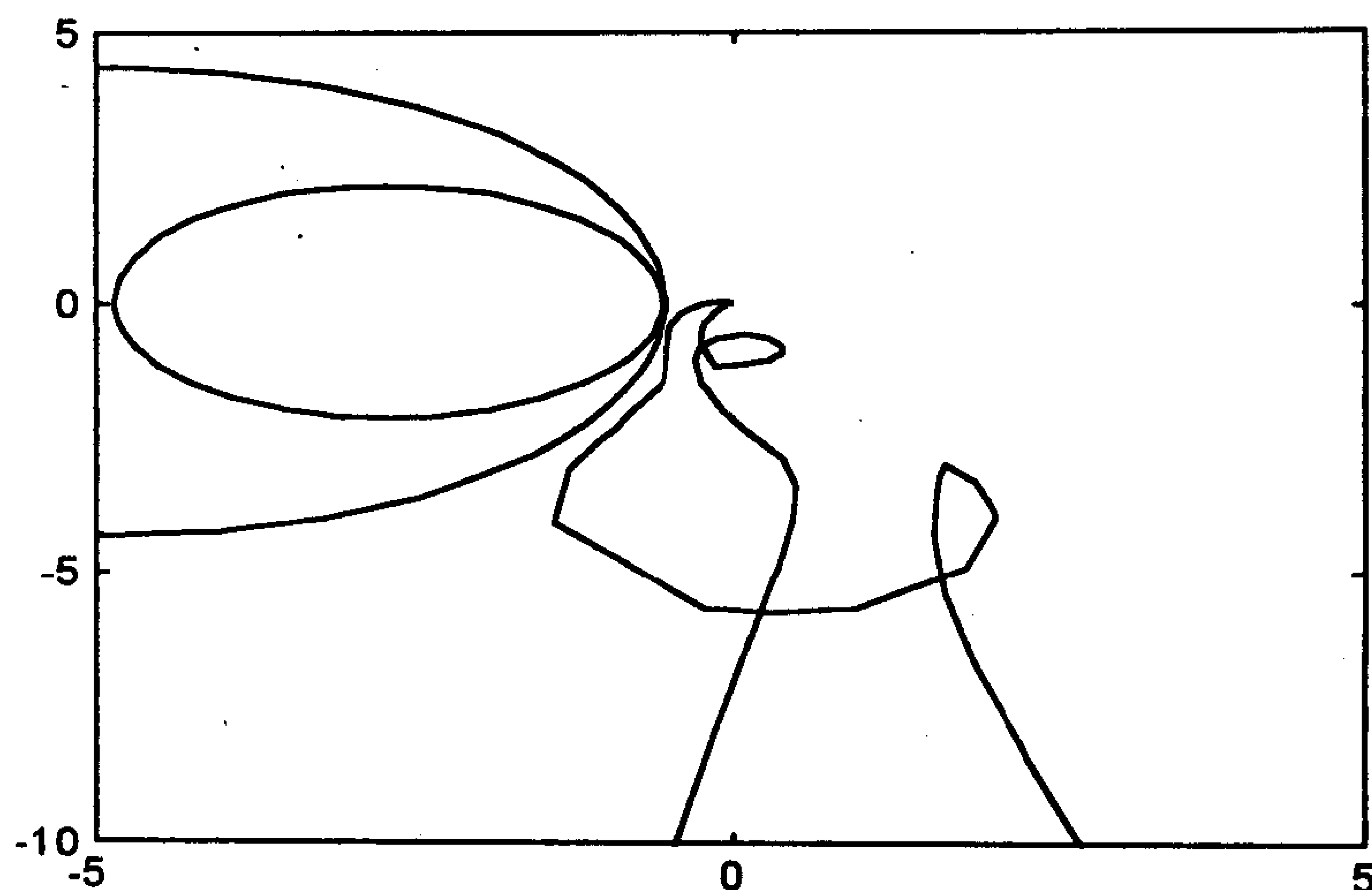


图6-25 最终系统的特征轨迹曲线

6.3.2 多变量系统的参数最优化设计

文献 [10] 提出了一种实用的参数最优化方法来设计多变量系统，假定该系统的框图如图 6-26 所示，其中和前面的叙述一致，$G(s)$ 为系统的对象传递函数矩阵，$K(s)$ 为

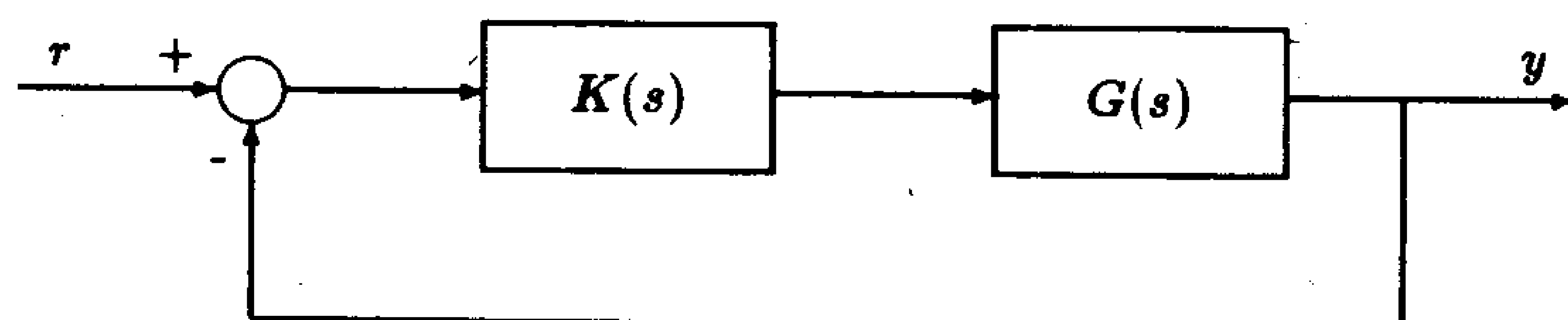


图6-26 多变量控制系统的结构图

控制器传递函数矩阵，且令输入和输出的个数分别为 l 和 m。这时系统的闭环传递函数矩阵可以写成

$$T(s) = G(s)K(s)[I + G(s)K(s)]^{-1} \tag{6.3.4}$$

若在某一个给定的频率区域内，能使得闭环传递函数矩阵尽可能地接近于一个预先指定的目标传递函数矩阵，则就能以某种参数最优化的方法来设计出合适的控制器，这就是文献 [10] 的最初目的。假设目标传递函数矩阵可以表示为 $T_t(s)$ (为推导方便以后省去 (s) 字样)，则对应于 T_t 的目标控制器 K_t 满足 $GK_t = T_t(I - T_t)^{-1}$。从而定义一个误差函数 $E = T_t - T$，则通过简单的变换可以很容易地证明

$$E = (I - T)(GK - GK_t)(I - T_t) \tag{6.3.5}$$

若 $\|E\|$ 足够小，亦即使得 K 足够接近 K_t, 则可以得出

$$E = (I - T_t)(GK - GK_t)(I - T_t) + O(\|E\|^2) \simeq (I - T_t)(GK - GK_t)(I - T_t) \qquad (6.3.6)$$

定义 $K(s) = N(s)/d(s)$, 其中 $d(s)$ 为用户选定的公分母多项式，且 $N(s)$ 为已知阶次的多项式矩阵，但其参数为待定的，定义 $B = I - T_t$, $A = BG/d(s)$, 且 $Y = BGK_tB$, 则式 (6.3.6) 可以写成

$$Y(s) \simeq A(s)N(s)B(s) + E(s) \qquad (6.3.7)$$

为找出最优的 $N(s)$ 参数，则可以定义下面的最优化准则

$$\|E\|_2^2 = \int_{-\infty}^{\infty} \mathrm{tr}\left[E^T(-j\omega)E(j\omega)\right]\mathrm{d}\omega \qquad (6.3.8)$$

式中

$$Y(s) = [y_1(s), \cdots, y_m(s)], N(s) = [n_1(s), \cdots, n_m(s)], E(s) = [e_1(s), \cdots, e_m(s)] \qquad (6.3.9)$$

这样可以写出下面的关系式

$$\begin{bmatrix} y_1(s) \\ y_2(s) \\ \vdots \\ y_m(s) \end{bmatrix} \simeq \left[B^T(s) \otimes A(s)\right] \begin{bmatrix} n_1(s) \\ n_2(s) \\ \vdots \\ n_m(s) \end{bmatrix} + \begin{bmatrix} e1(s) \\ e2(s) \\ \vdots \\ e_m(s) \end{bmatrix} \qquad (6.3.10)$$

控制器分子多项式 $n_i(s)$ 可以写成 $n_i(s) = [n_{1i}(s), n_{2i}(s), \cdots, n_{li}(s)]^T$, 且假设

$$n_{ij}(s) = v_{ij}^0 s^p + v_{ij}^1 s^{p-1} + \cdots + v_{ij}^{p-1} s + v_{ij}^p \qquad (6.3.11)$$

其中对 $n_i(s)$ 来说，p 可以是一个选定的正整数，这样可以用矩阵的形式来描述分子系数，对某些阶次较低的子多项式仍可作这样的设置，只不过令其高次项的系数等于 0 就可以了。构造如下矩阵

$$\Sigma(s) = \begin{bmatrix} s^p & s^{p-1} & \cdots & 1 & & & & & & & & \\ & & & & s^p & s^{p-1} & \cdots & 1 & & & & \\ & & & & & & & & \ddots & & & \\ & & & & & & & & & s^p & s^{p-1} & \cdots & 1 \end{bmatrix} \qquad (6.3.12)$$

则

$$\begin{bmatrix} n_1(s) \\ \vdots \\ n_m(s) \end{bmatrix} = \Sigma v, \text{ 且 } v = [v_{11}^0, v_{11}^1, \cdots, v_{ml}^p]^T \qquad (6.3.13)$$

若令 $X(s) = [B^T(s) \otimes A(s)]\Sigma(s), \eta(s) = [y_1^T(s), \cdots, y_m^T(s)]^T, \varepsilon(s) = [e_1^T(s), \cdots, e_m^T(s)]^T$, 则式 (6.3.10) 将写成下面的最小二乘标准形式

$$\eta(s) = X(s)v + \varepsilon(s) \qquad (6.3.14)$$

为得出 η 和 X 矩阵，则可以通过频率分析的方法获得在一些选定频率点 $\{\omega_i\}, i=1,\cdots,M$，通过前面的各式近似出 $X(j\omega_i)$ 和 $\eta(j\omega_i)$，从而构成 $X(j\omega)$ 和 $\eta(j\omega)$ 矩阵

$$X(j\omega)=\begin{bmatrix} X(j\omega_1) \\ X(j\omega_2 \\ \vdots \\ X(j\omega_M) \end{bmatrix}, \quad \eta(j\omega)=\begin{bmatrix} \eta(j\omega_1) \\ \eta(j\omega_2) \\ \vdots \\ \eta(j\omega_M) \end{bmatrix} \tag{6.3.15}$$

显然由式 (6.3.14) 得出问题的最小二乘解

$$\hat{v}=\left[X^T(-j\omega)X(j\omega)\right]^{-1}X^T(-j\omega)\eta(j\omega) \tag{6.3.16}$$

仔细观察上式立即可以发现问题：由此式得出的 $\hat{v}$ 参数难免出现复数值，这就使得得出的控制器无法实现，所以要对上面的计算方式进行改进，以确保得出实数的 v 值。文献 [10] 给出了这样的算法

$$\hat{v}=\Re e\left[X^T(-j\omega)X(j\omega)\right]^{-1}\Re e\left[X^T(-j\omega)\eta(j\omega)\right] \tag{6.3.17}$$

虽然可以这样一次性地得出控制器的参数 $\hat{v}$，但其结果究竟如何还应该放到闭环系统中去检验，这就需要下面的迭代过程来找到一个合适的 $\hat{v}$ 向量，从而最终设计出可用的控制器来。假设从式 (6.3.17) 可以设计出控制器 K_i，并可以通过下式构造出下一个控制器 $K_{i+1}=K_i+(\Delta K)_i$，且由这样两个控制器分别可以得出闭环传递函数矩阵 T_i 和 T_{i+1}，令误差量 $E_i=T_t-T_i$，则可以证明

$$E_i=(I-T_i)G(K_t-K_i)(I-T_t), \quad E_{i+1}=(I-T_{i+1})G(K_t-K_{i+1})(I-T_t) \tag{6.3.18}$$

由这样的迭代公式就可以得出使得 $\|E_{i+1}\|_2$ 最小化的控制器参数 K_{i+1} 来。

例 6.8 考虑下面给出的一个状态方程模型

$$A=\begin{bmatrix} 0 & 0 & 1.1320 & 0 & -1 \\ 0 & -0.0538 & -0.1712 & 0 & 0.0705 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.0485 & 0 & -0.8556 & -1.013 \\ 0 & -0.2909 & 0 & 1.0532 & -0.6859 \end{bmatrix}, B=\begin{bmatrix} 0 & 0 & 0 \\ -0.120 & 1 & 0 \\ 0 & 0 & 0 \\ 4.419 & 0 & -1.665 \\ 1.575 & 0 & -0.0732 \end{bmatrix}, C=\begin{bmatrix} I_3 & 0_{2\times 2} \end{bmatrix}$$

可见此系统为 3 输入 3 输出的模型，对此系统选择闭环目标传递函数为

$$T_t(s)=\mathrm{diag}\left\{\frac{3^2}{(s+3)^2}, \frac{3^2}{(s+3)^2}, \frac{10^2}{(s+10)^2}\right\}$$

则根据 T_t 可以由下面命令

```
>> A=[0,0,1.1320,0,-1; 0,-0.0538,-0.1712,0,0.0705; 0,00,0,1,0;
      0,0.0485,0,-0.8556,-1.013;0,-0.2909,0,1.0532,-0.6859];
>> B=[0,0,0; -0.120,1,0; 0,0,0; 4.419,0,-1.665; 1.575,0,-0.0732]; p=3; q=3;
>> C=[eye(p) zeros(p,2)]; D=zeros(p,p); w=logspace(-3,2);
>> Gg=mv2fr(A,B,C,D,w); iG=finv(w,Gg);
```

```
>> dx1=conv([1,3],[1,3]); dx2=conv([1,10],[1,10]); den_T=conv(dx1,dx2);
>> num_T=[0,0,3^2*dx2, 0,0,0,0,0, 0,0,0,0,0;
         0,0,0,0,0, 0,0,3^2*dx2, 0,0,0,0,0;
         0,0,0,0,0, 0,0,0,0,0, 0,0,10^2*dx1];
>> Gt=mv2fr(num_T,den_T,w); num_iT=num_T;
>> for i=1:p, num_iT(i,(i-1)*5+1:i*5)=den_T-num_T(i,(i-1)*5+1:i*5); end
>> GiT=mv2fr(num_iT,den_T,w); iGiT=finv(w,GiT); GG=[]; GG1=[];
>> for i=1:length(w), GG=[GG; iG(p*(i-1)+1:p*i,:)*Gt(p*(i-1)+1:p*i,:)]; end
>> for i=1:length(w), GG1=[GG1; GG(p*(i-1)+1:p*i,:)*iGiT(p*(i-1)+1:p*i,:)]; end
>> for i=1:p, for j=1:q
      subplot(p,q,p*(i-1)+j); iG=fget(w,GG1,[i,j]); semilogx(w,20*log10(abs(iG)))
   end, end
```

绘制出目标控制器 $K_t = G^{-1}T_t(I-T_t)^{-1}$ 的 Bode 图形，如图 6-27 所示。由该图可见，对第 1 输入

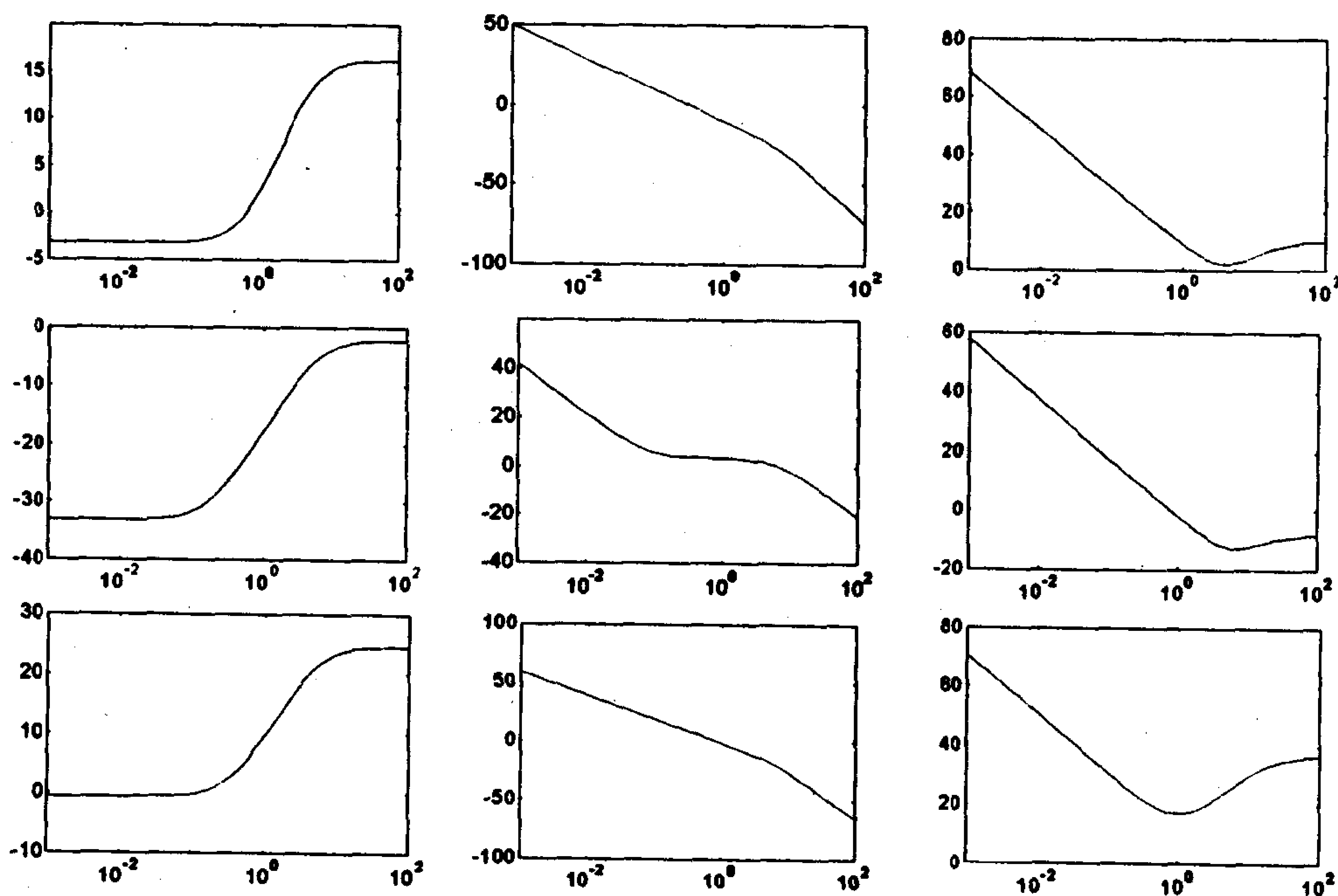


图6-27 目标控制器的 Bode 图形

并不需要加一个积分器，而其它两个输入需要加一个积分器。另外由图可以看出，对第 1 输入选择 −6 作为极点位置比较合适，而对第 2, 3 输入可以分别选择 −6 和 −30 作极点位置，这样可以按下面方式设置控制器

$$k_{i1} = \frac{v_{i1}^0 s + v_{i1}^1}{s+6},\ k_{i2}(s) = \frac{v_{i2}^0 s^2 + v_{i2}^1 s + v_{i2}^2}{s(s+6)},\ k_{i3}(s) = \frac{v_{i3}^0 s^2 + v_{i3}^1 s + v_{i3}^2}{s(s+30)}$$

这样通过最优化算法可以得出如表 6-1 所示的控制器参数，对此系统进行仿真，则可以得出如图 6-28 所示的阶跃响应输出曲线。由最终仿真结果可以看出，在设计出来的控制器下系统的闭环响应和指定的目标模型几乎是完全一致的，由此可以得出结论，由参数最优化算法设计出来的控制器可以有很简单的结构，而其控制效果是相当完美的，完全可以和具有复杂控制器结构的 LQG 及 H_∞ 相媲美[24]。

表6-1　参数最优化得出的控制器参数表

i \ j	1	2	3
1	$v_{11}^0=-6.5183$ $v_{11}^1=-4.1806$	$v_{12}^0=8\times10^{-16}$ $v_{12}^1=6\times10^{-15}$ $v_{12}^2=1.9101$	$v_{13}^0=-5.2967$ $v_{13}^1=6.5509$ $v_{13}^2=77.9300$
2	$v_{21}^0=-0.7822$ $v_{21}^1=0.1328$	$v_{22}^0=9\times10^{-15}$ $v_{22}^1=9.0000$ $v_{22}^2=0.7134$	$v_{23}^0=-0.6153$ $v_{23}^1=0.6702$ $v_{23}^2=22.989$
3	$v_{31}^0=-17.300$ $v_{31}^1=-5.6199$	$v_{32}^0=-1\times10^{-15}$ $v_{32}^1=9\times10^{-15}$ $v_{32}^2=5.3316$	$v_{33}^0=-99.88$ $v_{33}^1=-62.41$ $v_{33}^2=104.81$

图6-28　多变量系统阶跃响应曲线

6.4　自整定 PID 控制策略

PID 控制是最早发展起来的控制策略之一[7]，因为它所涉及的设计算法和控制结构都是很简单的，并且十分适用于工程应用背景，此外 PID 控制方案并不要求精确的受控对象的数学模型，且采用 PID 控制的控制效果一般是比较令人满意的，所以工业界实际应用中 PID 控制器是应用最广泛的一种控制策略，且都是比较成功的。近年来，在控制理论研究和实际应用中 PID 又重新引起人们的注意，这是因为瑞典学者 Karl Åström 等人推出的智能型 PID 自整定控制器表现出了传统 PID 难以实现的控制性能[3]，并出现了自整定 PID 控制器的硬件商品，使得 PID 控制更广泛地应用于工业控制中。

6.4.1 Ziegler-Nichols 经验公式

如果实际要求不是特别高，则一般过程控制问题都可以用 PID 型控制器比较好地解决 [2] 。所谓 PID 控制器，就是对误差信号进行加权的比例、积分与微分运算，最后将其和送给受控对象，以完成整个控制过程。传统的 PID 控制策略可以参见图 6-29 中给出的控制结构， 这里标准的 PID 控制器模型为

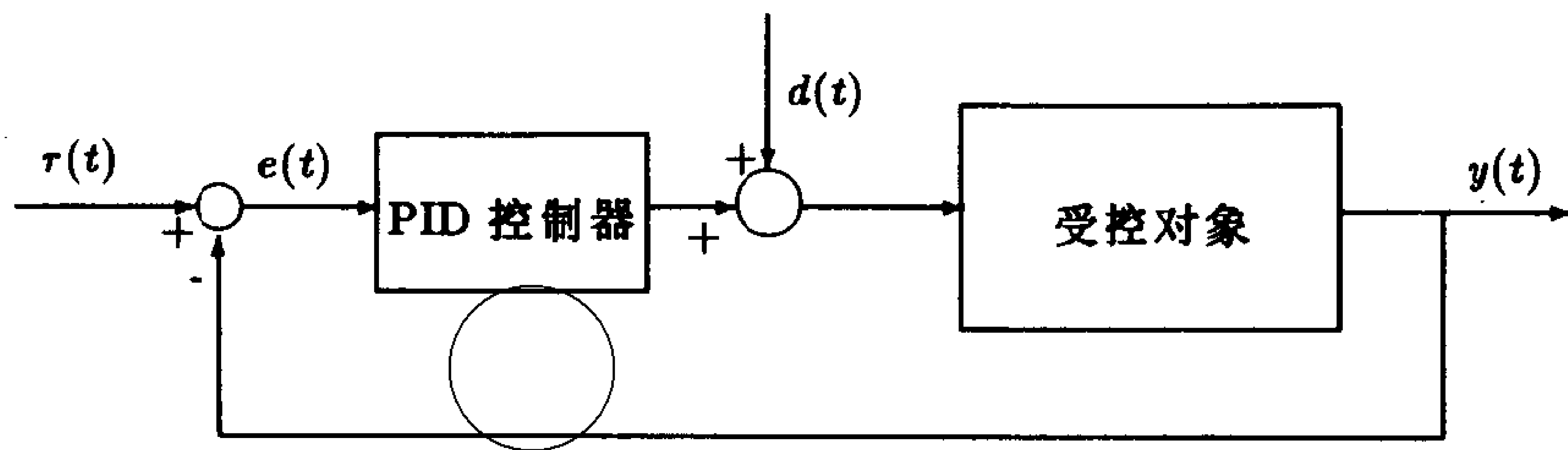


图6-29　典型 PID 控制器结构

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)\mathrm{d}\tau + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right] \tag{6.4.1}$$

式中 $u(t)$ 为进入受控对象的控制变量， $e(t) = u_c - y(t)$ 为误差信号，而 u_c 为给定参考输入的值。然而在实际应用中，人们往往要对该控制结构进行改动，例如其中的纯微分环节往往由一个带惯性的一阶环节所取代

$$U(s) = K_p\left(1 + \frac{1}{T_i s} + \frac{sT_d}{1+sT_d/N}\right)E(s) \tag{6.4.2}$$

并通常将 N 取作 10 。

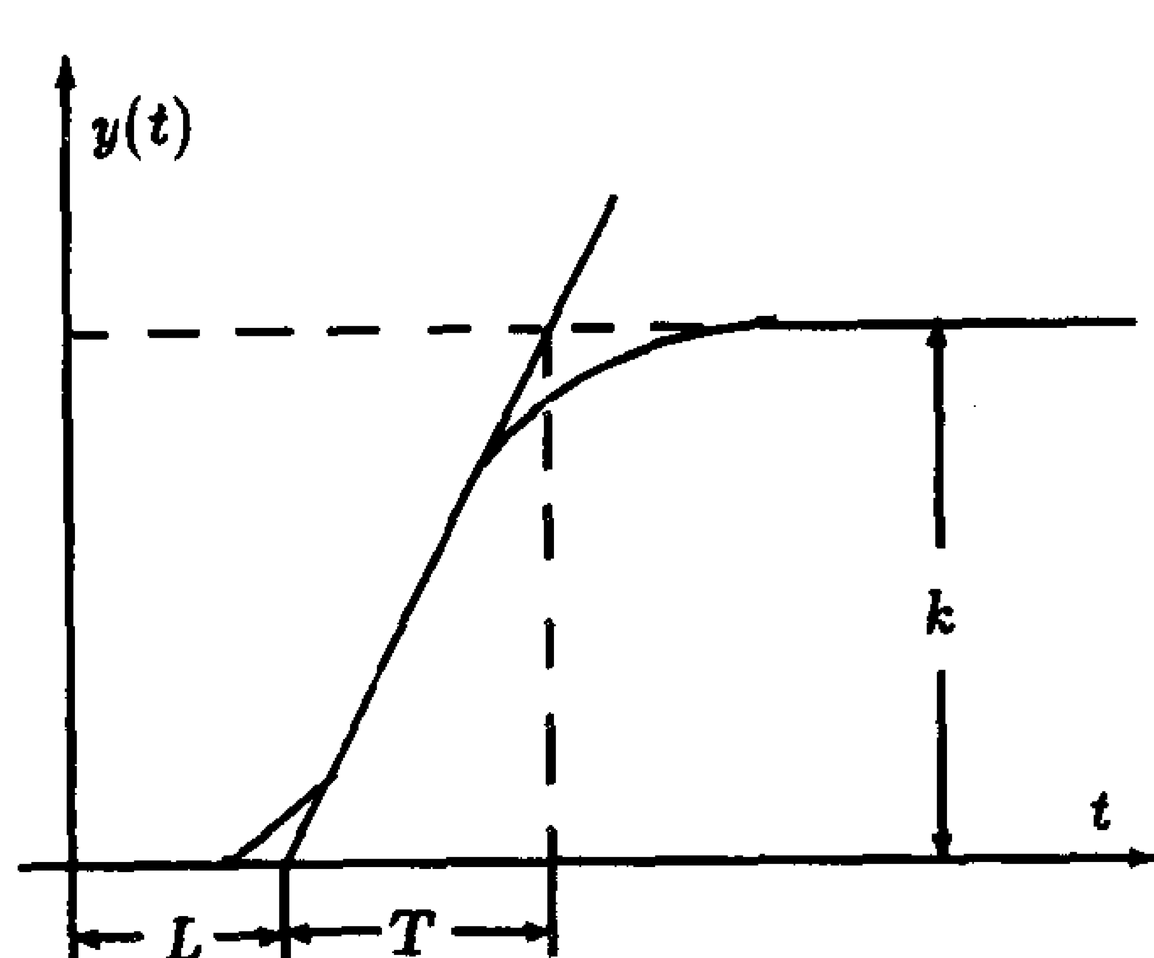
Ziegler 与 Nichols 提出了调节 PID 控制器参数的经验公式 [38]，这一调节器可以分别对带有纯时间延迟的一阶近似模型或频率响应数据来设定。假设控制系统接近于一阶带有延迟环节的模型

$$G(s) = \frac{k}{1+sT}\mathrm{e}^{-sL} \tag{6.4.3}$$

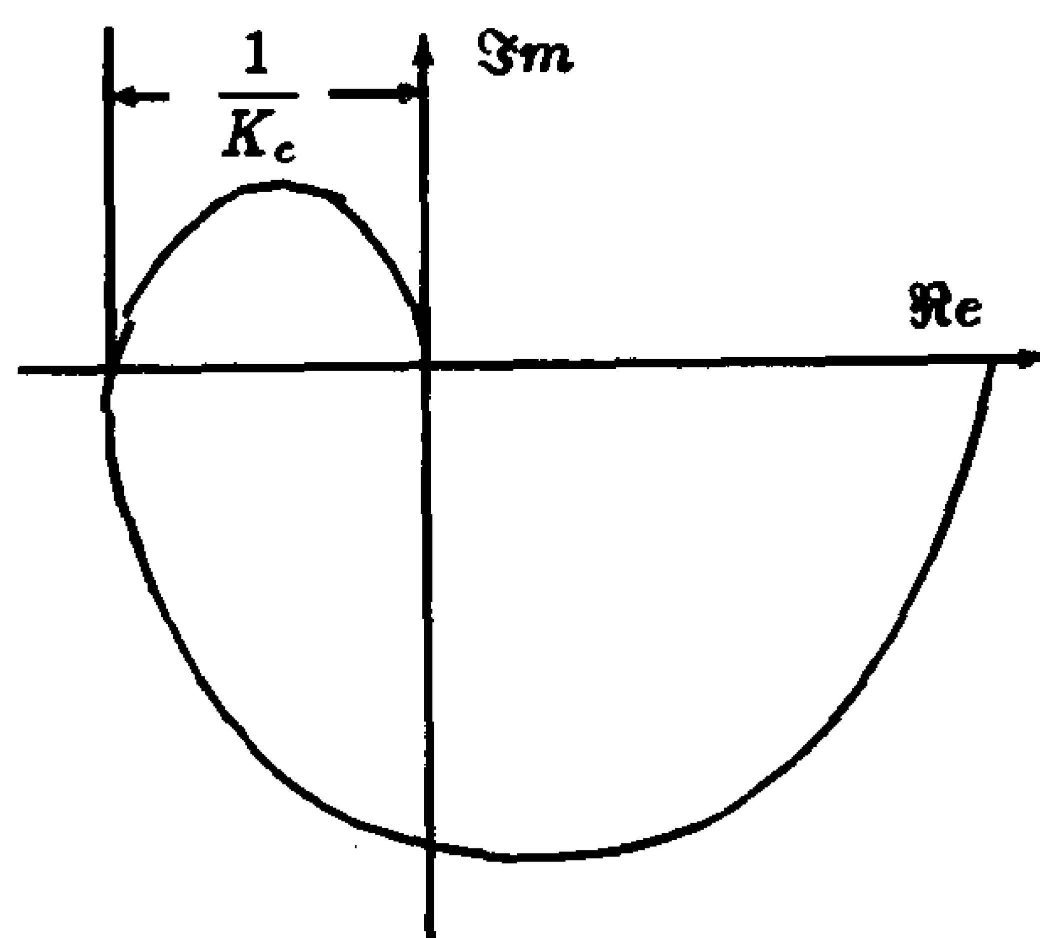
其中一阶响应的特征参数 k, L 和 T 可以由图 6-30 (a) 构成的示意图提取出来，求取这些参数对实际系统并不困难，首先对该系统进行阶跃输入激励， 得出响应曲线，再根据曲线得出这些特征参数。有了这些参数，并假设 $a = kL/T$, 则 P (比例控制器) 、PI (比例加积分控制器) 与 PID 控制器的参数可以由表 6-2 求出来。

若已知系统的频率响应数据，即从 Nyquist 图形上直接得出剪切频率 ω_c 和该点处的增益值 K_c, 如图 6-30(b) 所示，这样就可以得出 $T_c = 2\pi/\omega_c$, 这时 P, PI 及 PID 控制器的参数也可以由表 6-2 求出。

除了标准的 Ziegler-Nichols 设定方法之外，还有很多其它的设定方法，如改进的 Ziegler-Nichols 方法 [13] 、幅值相位裕度设定方法 [1] 及最优整定方法 [37] 等，其目的都是提供简单地调整参数的方法，以达到较好的控制效果，下面将分别介绍各种参数设定方法：



(a). 阶跃响应示意图



(b). 频率响应示意图

图6-30 响应曲线示意图

表6-2 Ziegler-Nichols 设定算法

控制器类型	由模型设定			由频率响应设定		
	K_p	T_i	T_d	K_p	T_i	T_d
P	$1/a$			$0.5K_c$		
PI	$0.9a$	$3L$		$0.4K_c$	$0.8T_c$	
PID	$1.2/a$	$2L$	$L/2$	$0.6K_c$	$0.5T_c$	$0.12T_c$

- **改进的 Ziegler-Nichols 整定方法**: 标准的 Ziegler-Nichols 设定方法设计出来的 PID 控制器在设定点响应中经常会得出很强的振荡曲线，且往往其超调量很大，文献 [13] 对比例的分量进行相应的调节，给出了如下的 PID 控制器结构

$$u(t) = K_p\left[(\beta u_c - y) + \frac{1}{T_i}\int e\mathrm{d}t - T_d\frac{dy}{\mathrm{d}t}\right] \tag{6.4.4}$$

该方案把微分动作放到输出信号处去完成，并对比例输入部分进行了修正。本方法引入了规范化的死区时间常数 τ 和规范化的一阶时间常数 κ, 对照式 (6.4.3) 给出的一阶模型，这些规范化参数可以如下定义

$$\kappa = K_c k,\quad \tau = \frac{L}{T},\quad \text{且满足}\quad \kappa = 2\left(\frac{11\tau+13}{37\tau-4}\right) \tag{6.4.5}$$

对不同的 κ 或 τ 所在的范围，可以按照下面的方式来求出 β 的值，并可以根据需要对传统 Ziegler-Nichols 参数作出适当的修正

* 若 $2.25 < \kappa < 15$ 或 $0.16 < \tau < 0.57$, 则应该保持 Ziegler-Nichols 参数，并为使超调量小于 10% 或 20% 分别如下引入 β 系数

$$\beta = \frac{15-\kappa}{15+\kappa},\quad \beta = \frac{36}{27+5\kappa} \tag{6.4.6}$$

* 若 $1.5 < \kappa < 2.25$ 或 $0.57 < \tau < 0.96$, 则应该将 Ziegler-Nichols 积分系数修正为 $T_i = 0.5\mu T_c$, 其中

$$\mu = \frac{4}{9}\kappa,\quad \beta = \frac{8}{17}(\mu - 1) \tag{6.4.7}$$

* 若 $1.2 < \kappa < 1.5$, 则为使超调量小于 10%, 应如下修正 PID 系数

$$K_p = \frac{5}{6}\left(\frac{12+\kappa}{15+14\kappa}\right),\quad T_i = \frac{1}{5}\left(\frac{4}{15}\kappa + 1\right) \tag{6.4.8}$$

- **基于幅值相位裕度指定的参数设定方法:** Åström-Hägglund 引入的一种设定方法是想在某一个频率点处，将最终系统的频率特性值强制地移动到某一个特定点，例如使得该点的幅值为 1，而相位为一个预先指定的值，或将相位值设置成 $-180°$, 而将幅值设置成预先指定的值，从而强制地使得闭环系统幅值或相位裕度达到指定的要求。假设在某一频率点处受控对象和控制器的频率响应值分别可以写成

$$G_p(j\omega) = r_p e^{j(\pi+\phi_p)},\quad G_c(j\omega) = r_c e^{j\phi_c} \tag{6.4.9}$$

而期望的频率响应为 $B = r_s e^{j(\pi+\phi_s)}$, 这时有

$$r_c r_p e^{j(\pi+\phi_c+\phi_p)} = r_s e^{j(\pi+\phi_s)} \tag{6.4.10}$$

则可以按照下面的两种情形分别进行设计

* 若按照指定幅值裕度来设计，则可以指定 $\phi_s = 0$, 且 $r_s = 1/A_m$, 其中 A_m 为指定的幅值裕度，这时控制器的幅值和相位分别为

$$r_c = \frac{r_s}{r_p} = \frac{1}{A_m r_p},\quad \phi_c = \phi_s - \phi_p = -\phi_p \tag{6.4.11}$$

* 若按照指定的相位裕度来设计，则可以指定 $r_s = 1$, 且 $\phi_s = \phi_m$, 其中 ϕ_m 为指定的相位裕度，这时有

$$\omega_c T_d - \frac{1}{\omega_c T_i} = \tan\phi_m \tag{6.4.12}$$

这里 ω_c 为指定频率点，为了确定 T_i 和 T_d, 则可以指定二者满足某种线性关系 $T_i = \alpha T_d$, 这样可以解出 T_d 的值

$$T_d = \frac{\tan\phi_m + \sqrt{4/\alpha + \tan^2\phi_m}}{2\omega_c} \tag{6.4.13}$$

此外还可以根据下式设计出 K_p 参数

$$K_p = \frac{\cos\phi_m}{|G(j\omega_c)|} = K_c\cos\phi_m \tag{6.4.14}$$

- **PID控制器的最优设定方法**: 庄敏霞与 Atherton 针对各种指标函数得出了最优 PID 参数指定的算法 [36, 37], 考虑下面给出的最优指标通式

$$J_n(\theta) = \int_0^\infty [t^n e(\theta, t)]^2 \mathrm{d}t \tag{6.4.15}$$

这里 $e(t)$ 为进入 PID 控制器的误差信号，参见图 6-29 中的系统结构，该算法重点讨论了两种输入下的 PID 控制器的自整定方法，即根据设定点 (set-point) 信号的最优自整定算法和基于扰动信号 $d(t)$ 的最优自整定算法。对式 (6.4.15) 中给出的最优指标着重考虑 3 种情况，即 $n = 0$, 简记作 ISE (integral squared error) 准则， $n = 1$, 简记为 ISTE 准则； $n = 2$, 简记为 IST^{2}E 准则。在最优自整定算法中分别分下面 3 种情况加以考虑：

* 若已知系统的数学模型如式 (6.4.3) 给出，则对典型 PID 结构可以建立经验公式

$$K_p = \frac{a_1}{k}\left(\frac{L}{T}\right)^{b_1}, \quad T_i = \frac{T}{a_2 + b_2(L/T)}, \quad T_d = a_3 T\left(\frac{L}{T}\right)^{b_3} \tag{6.4.16}$$

对不同的 L/T 范围，可以得出 (a,b) 参数表如表 6-3 所示。由表中给出的 PID 参

表6-3 设定点PID控制器参数表

L/T范围	0.1 − 1			1.1 − 2		
准　则	ISE	ISTE	IST^{2}E	ISE	ISTE	IST^{2}E
a_1	1.048	1.042	0.968	1.154	1.142	1.061
b_1	-0.897	-0.897	-0.904	-0.567	-0.579	-0.583
a_2	1.195	0.987	0.977	1.047	0.919	0.892
b_2	-0.368	-0.238	-0.253	-0.220	-0.172	-0.165
a_3	0.489	0.385	0.316	0.490	0.384	0.315
b_3	0.888	0.906	0.892	0.708	0.839	0.832

数设置可以通过 MATLAB 来简单地实现，例如可以编写出一个函数来计算 PID 控制器参数，在调用此函数时应该首先指定参数 T, L 和 k, 然后指定最优准则类型，如 key=1 则表示 ISE 准则， key=2 表示 ISTE 准则， key=3 表示 IST^{2}E 准则，这样就可以写出下面的 MATLAB 函数

```
function [Kp, Ti, Td]=getPID1(T,L,k,key)
PIDtab=[ 1.048, 1.042, 0.968, 1.154, 1.142, 1.061;
        -0.897,-0.897,-0.904,-0.567,-0.579,-0.583;
         1.195, 0.987, 0.977, 1.047, 0.919, 0.892;
        -0.368,-0.238,-0.253,-0.220,-0.172,-0.165;
         0.489, 0.385, 0.316, 0.490, 0.384, 0.315;
         0.888, 0.906, 0.892, 0.708, 0.839, 0.832];
```

```
ii=0; if (L/T>1) ii=3; end; tt=L/T;
a1=PIDtab(1,ii+key); b1=PIDtab(2,ii+key); a2=PIDtab(3,ii+key);
b2=PIDtab(4,ii+key); a3=PIDtab(5,ii+key); b3=PIDtab(6,ii+key);
Kp=a1/k*tt^b1;
Ti=T/(a2+b2*tt);
Td=a3*T*tt^b3;
```

在很多情况下，往往希望在反馈回路实现微分控制的动作，而不希望将之放在前向回路，这时 PID 控制器可以写成

$$U(s)=K_p\left(1+\frac{1}{T_i s}\right)E(s)-\frac{sT_d}{1+sT_d/N}Y(s) \tag{6.4.17}$$

这时控制器参数仍满足式 (6.4.16), 但其 (a,b) 参数表如表 6-4 所示。

表6-4 设定点 PID 控制器参数表

L/T范围	0.1 − 1			1.1 − 2		
准　　则	ISE	ISTE	IST²E	ISE	ISTE	IST²E
a_1	1,260	1.053	0.942	1.295	1.120	1.001
b_1	-0.887	-0.930	-0.933	-0.619	-0.625	-0.624
a_2	0.701	0.736	0.770	0.661	0.720	0.754
b_2	-0.147	-0.126	-0.130	-0.110	-0.114	-0.116
a_3	0.375	0.349	0.308	0.378	0.350	0.308
b_3	0.886	0.907	0.897	0.756	0.811	0.813

在一些特殊情况下往往还会要求将控制器设置成抗干扰型的，即对干扰信号 $d(t)$ 有良好的抑制作用，这时控制器参数可以由下式得出

$$K_p=\frac{a_1}{T}\left(\frac{L}{T}\right)^{b_1},\quad \frac{1}{T_i}=\frac{a_2}{T}\left(\frac{L}{T}\right)^{b_2},\quad T_d=a_3T\left(\frac{L}{T}\right)^{b_3} \tag{6.4.18}$$

而控制器 (a,b) 参数可以由表 6-5 查出。

文献 [37] 中还给出了各种情况下 PI 控制器的结构和参数表，在这里就不再引述了，用户可以自己查询。对于给定模型的系统，可以采用文献 [33] 给出的次最优降阶算法得出其带有纯时间延迟的一阶模型，然后依赖前面介绍的 PID 算法来得出合适的 PID 控制器。

* 若给出系统的频率响应参数，特别地若已知受控对象的剪切频率 ω_c 和此时的增益 K_c, 则可以构造 $T_c=2\pi/\omega_c$, 这时可以设计出 3 种情况下的 PID 控制器，如表 6-6 所示，其中 $\kappa=kK_c$ 为规范化增益。

当进行继电型自整定时，由得出的振荡频率 ω_0 和峰值幅度 a_0 可以构造出 $T_0=2\pi/\omega_0$, 且 $K_0=4h/(a_0\pi)$, 并假定 $\kappa_0=kK_0$, 则可以依照表 6-7 来设计出相应的 PID 控制器来。



表6-5 抗干扰PID控制器参数表

L/T范围	0.1 − 1			1.1 − 2		
准　　则	ISE	ISTE	$\mathrm{IST^2E}$	ISE	ISTE	$\mathrm{IST^2E}$
a_1	1.473	1.468	1.531	1.524	1.515	1.592
b_1	-0.970	-0.970	-0.960	-0.735	-0.730	-0.705
a_2	1.115	0.942	0.971	1.130	0.957	0.957
b_2	-0.753	-0.725	-0.746	-0.641	-0.598	-0.597
a_3	0.550	0.443	0.413	0.552	0.444	0.414
b_3	0.948	0.939	0.933	0.851	0.847	0.850

表6-6 ISTE准则下的PID参数公式1

PID	设　定　点　型	抗　干　扰　型	微　分　反　馈　型
K_p	$0.509K_c$	$\frac{4.434\kappa-0.966}{5.12\kappa+1.734}K_c$	$\frac{4.437\kappa-1.587}{8.024\kappa-1.435}K_c$
T_i	$0.051(3.302\kappa+1)T_c$	$\frac{1.751\kappa-0.612}{3.776\kappa+1.388}T_c$	$0.037(5.89\kappa+1)T_c$
T_d	$0.125T_c$	$0.144T_c$	$0.112T_c$

* 改进的幅值相位方法：利用幅值相位算法可以由下式构造出PID控制器

$$K_p=\frac{m\cos\phi}{|G(j\omega_c)|}=mK_c\cos\phi,\quad T_d=\frac{\tan\phi+\sqrt{4/\alpha+\tan^2\phi}}{2\omega_c},\quad T_i=\alpha T_d \tag{6.4.19}$$

其中 $\alpha=0.413(3.302\kappa+1)$ 或 $\alpha=1.687\kappa_0$，常数 ϕ 和 m 可以如下求出：由规范化的增益 κ 可以得出

$$\phi=33.8^\circ(1-0.97\mathrm{e}^{-0.45\kappa}),\quad m=0.614(1-0.233\mathrm{e}^{-0.347\kappa}) \tag{6.4.20}$$

表6-7 ISTE准则下的PID参数公式2

PID	设　定　点　型	抗　干　扰　型	微　分　反　馈　型
K_p	$0.604K_0$	$\frac{6.068\kappa_0-4.273}{5.758\kappa_0-1.058}K_0$	$\frac{2.354\kappa_0-0.696}{3.363\kappa_0+0.517}K_0$
T_i	$0.04(4.972\kappa_0+1)T_0$	$\frac{1.1622\kappa_0-0.748}{2.516\kappa_0-0.505}T_0$	$0.271\kappa_0T_0$
T_d	$0.130T_0$	$0.15T_0c$	$0.1162T_0c$

若得出的是自整定频率和增益，则可以进行如下的计算

$$\phi = 33.2°(1 - 1.38e^{-0.68\kappa_0}), \quad m = 0.613(1 - 0.262e^{-0.44\kappa_0}) \tag{6.4.21}$$

例 6.9 考虑下面给出的受控对象模型[37]

$$G(s) = \frac{e^{-0.5s}}{(s+1)^2}$$

如果想获得一阶近似，则可以得出 $G(s) \simeq e^{-0.99s}/(1.65s+1)$，当然也可以采用文献 [33] 中给出的方法进行近似，这样就可以设计出各种 PID 控制器，其参数如表 6-8 所示，由这些控制器得出的闭环阶跃响应曲线如图 6-31 所示。

表6-8 各种 PID 算法参数表

控 制 方 法	K_p	T_i	T_d	β
Ziegler-Nichols 方法	2.813	1.636	0.409	
改进 Ziegler-Nichols 方法	2.813	1.636	0.409	0.524
Åström-Hägglund 算法	3.125	2.514	0.629	
ISTE 最优设定方法	1.648	1.955	0.40	
幅值相位整定方法	2.387	2.729	0.397	

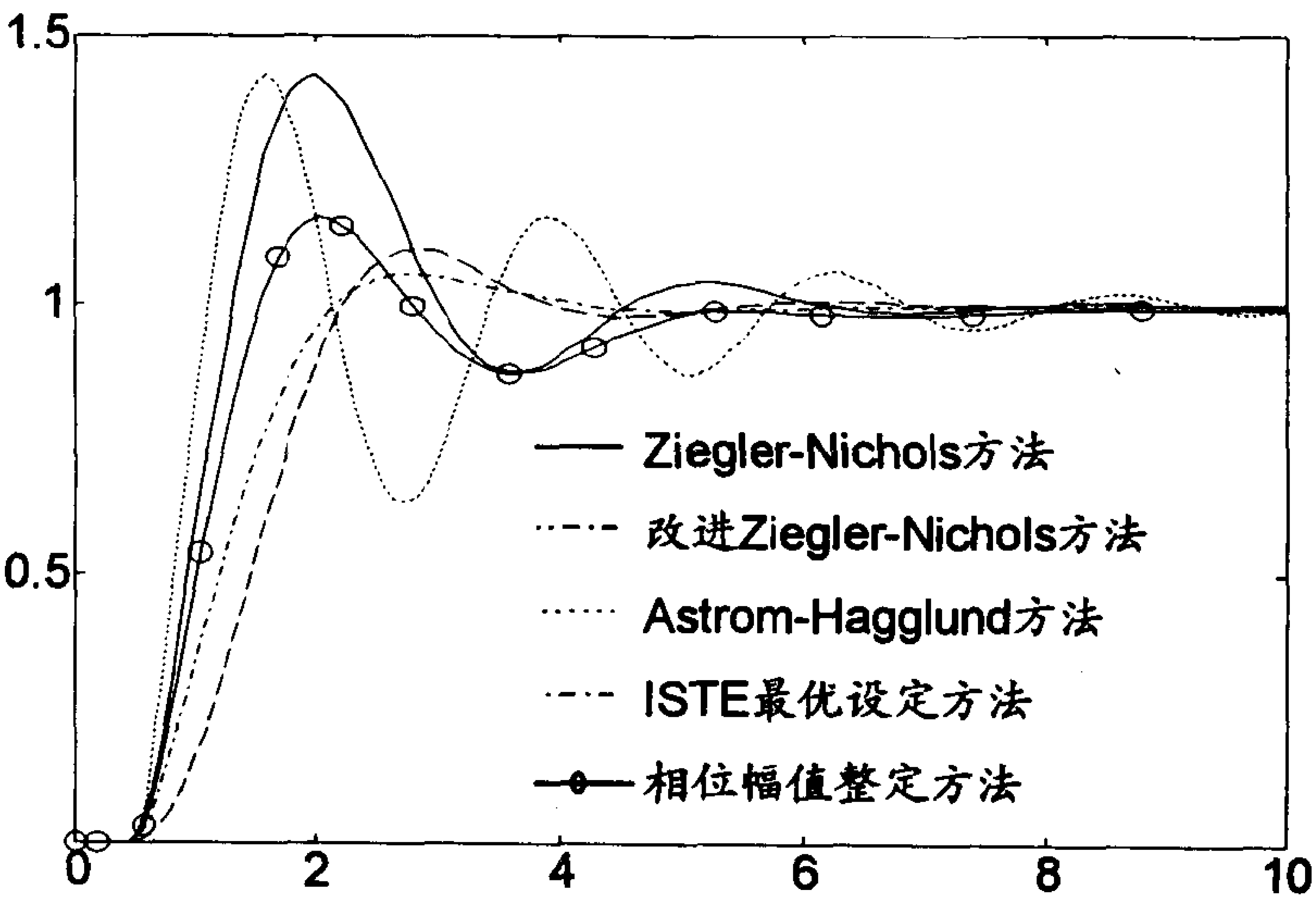


图6-31 各种 PID 算法的控制效果

例 6.10 若系统的模型为 $G(s) = 2/(s+1)^3$，则可以通过文献 [33] 中给出的次最优算法得出一阶降阶模型 $G_{0/1}(s) = 2e^{-1.3120s}/(3.0921s+1)$，并通过它设计出控制器参数 $K_p = 0.6165, T_i = 2.5041, T_d = 0.6424$，这样可以得出如图 6-32 所示的控制效果。

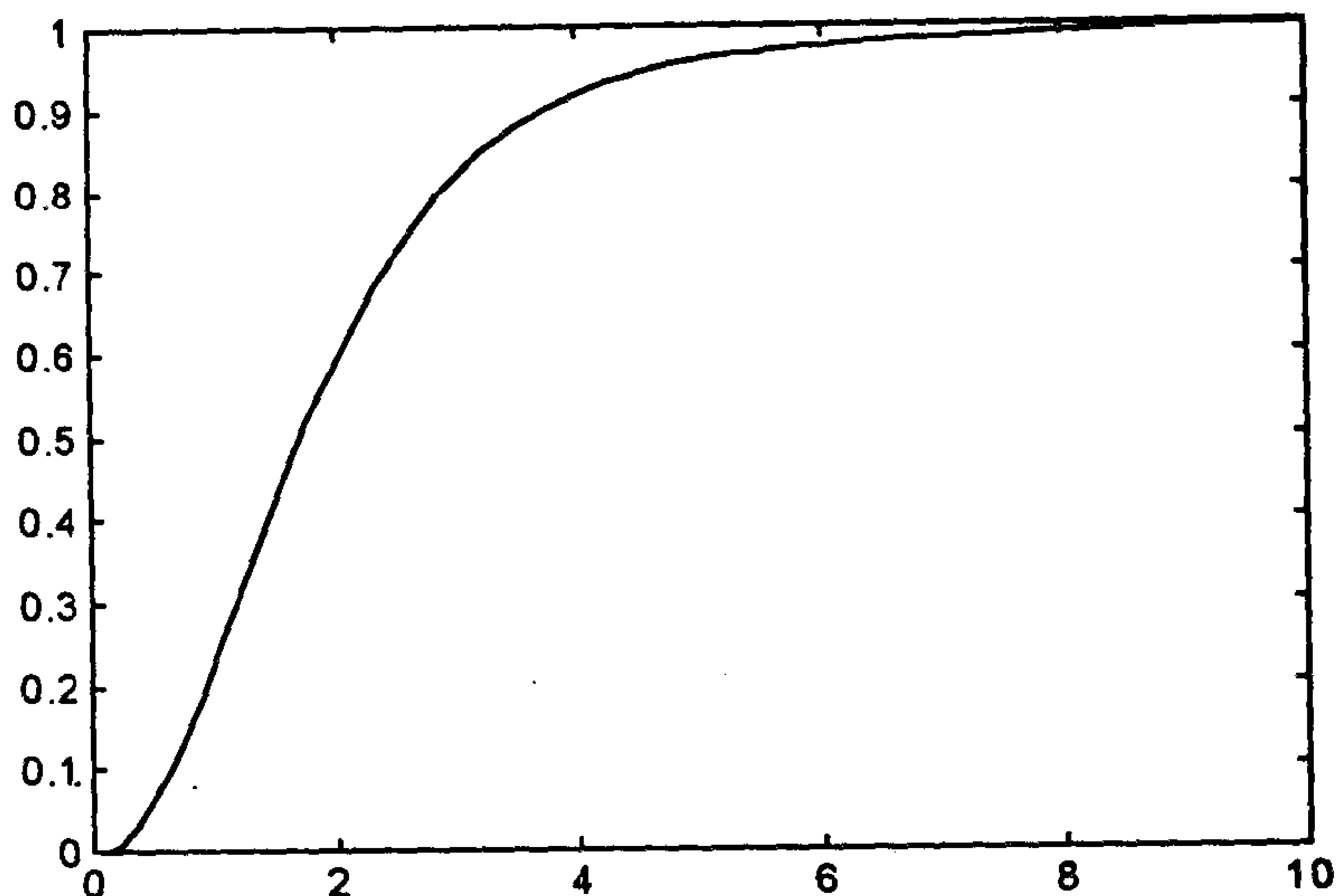


图6-32　3 阶系统的最优 PID 控制

6.4.2　PID 自整定控制结构与方法

由前面的叙述可知，若测出了系统的一阶模型 (6.4.3) 或得出了系统的振荡频率 ω_c 和增益 K_c, 则可以容易地设计出 PID 控制器。以往要想求出系统的这些特征参数，需要使用离线的方法来进行，即首先通过实验测出系统的特征参数，然后再根据这些参数设计一个合适的 PID 控制器，最后再将此控制器应用到原系统的控制中。若系统的参数发生变化，则应该再重新开始这一过程。

Åström 和 Hägglund 提出了一种称为自整定 (autotuning) 的方法来设计系统的 PID 控制器 [1], 该方案的基本想法是在控制系统中设置两种模态：测试模态和调节模态，在测试模态下由一个继电非线性环节来测试系统的振荡频率和增益，而在调节模态下由系统的特征参数首先得出 PID 控制器，然后由此控制器对系统的动态性能进行调节。如果系统的参数发生变化时，则需要重新进入测试模态进行测试，测试完成之后再回到调节模态进行控制。

继电型 PID 自整定控制结构如图 6-33 所示，从图中可以看出，两个模态之间的切换是靠开关来实现的。

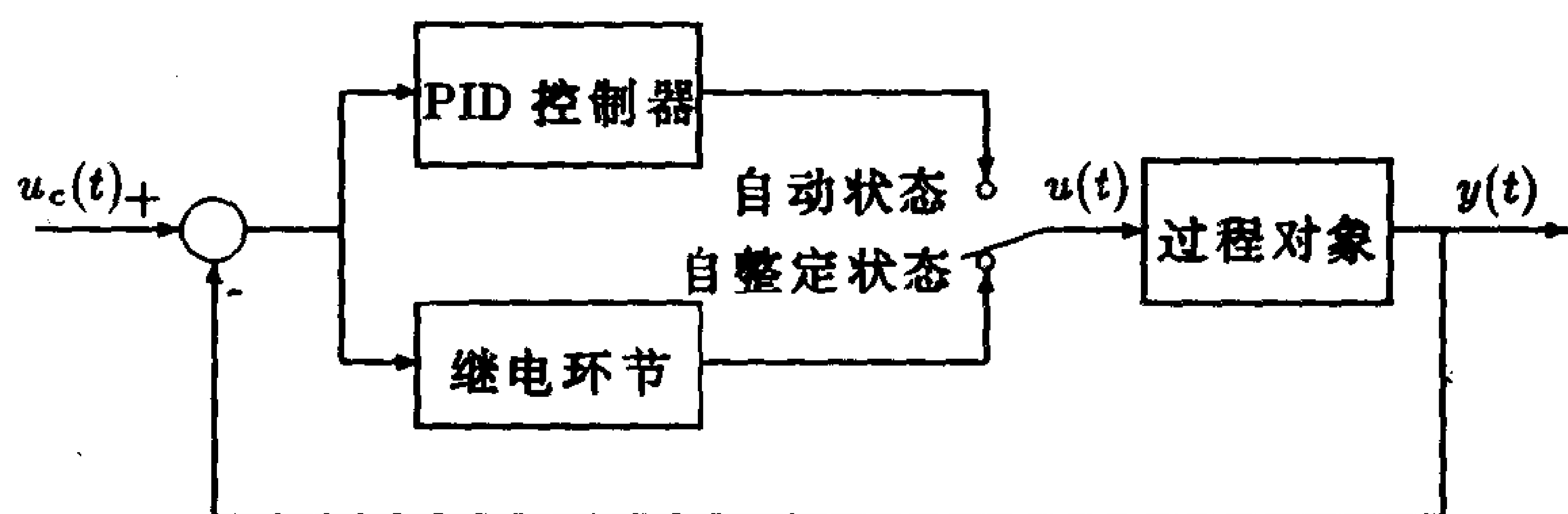


图6-33　继电型 PID 自整定控制结构

在测试模态下，系统的等效框图如图 6-34 (a) 所示，而系统的继电非线性环节如图 6-34 (b) 所示。确定系统的振荡频率 ω_c 与增益 K_c 有多种方法，比较常用的有描述函数

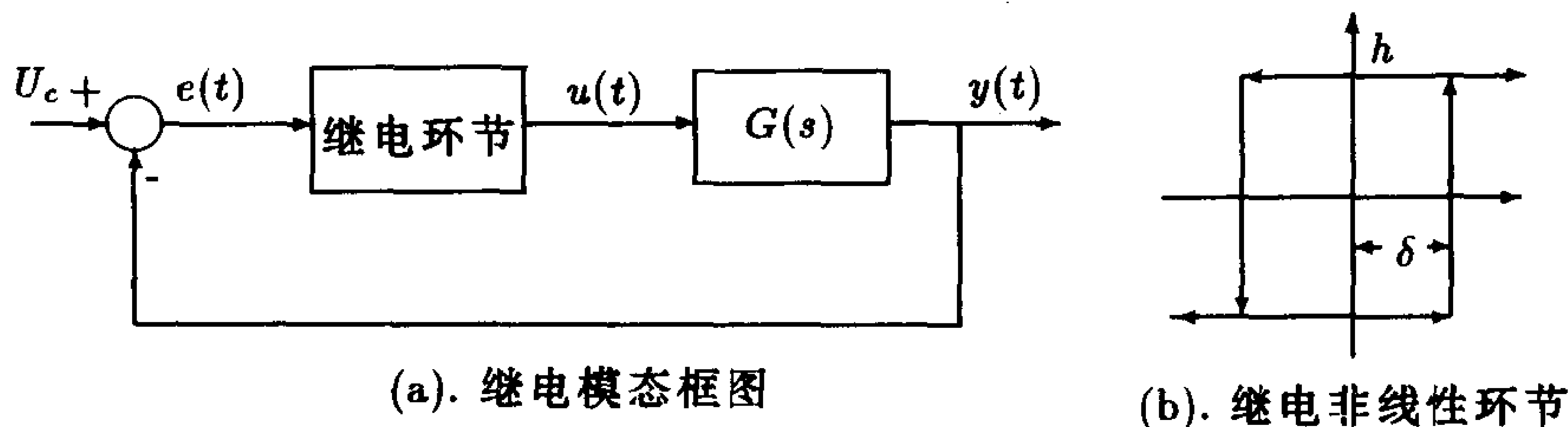


图6-34　继电控制非线性系统模型

方法 [5], 此方法是一种近似方法； Tsypkin 方法 [30] 对继电特性来说是一种精确方法，此外还可采用仿真的方法来求取 [36] 。

- **描述函数方法**: 描述函数方法实际上是根据非线性环节输入与输出信号之间的基波分量关系来进行近似的一种有效分析方法，由描述函数理论可知，图 6-34 (b) 中给出的带有回环的继电非线性环节的描述函数可以表述为

$$N(a)=\frac{4h}{\pi a^2}\left(\sqrt{a^2-\delta^2}-j\delta\right) \tag{6.4.22}$$

这时系统的闭环特征方程发生振荡的条件可以写成

$$1+N(a)G(s)\left.\right|_{s=j\omega_c}=0,\quad \text{即}\quad G(j\omega_c)=-\frac{1}{N(a)} \tag{6.4.23}$$

设该等式的实部和虚部均等于 0 则可以得出振荡频率 ω_c 和增益 K_c 。在这里只考虑一种简单的情形，假设继电非线性环节不带有回环，即若设 $\delta=0$, 则描述函数可以简化成 $N(a)=4h/(\pi a)$, 这时将立即求出系统的振荡频率和增益 K_c 。

$$K_c=\frac{4h}{\pi a},\quad T_c=\frac{2\pi}{\omega_c} \tag{6.4.24}$$

- **Tsypkin 方法**: Tsypkin 方法是专门针对继电系统的精确分析方法 [30], 它不仅考虑基波分量，而且考虑各次谐波分量。继电非线性有这样一个特点，其输出为方波信号，这样其 Fourier 级数可以写成

$$y(t)=\sum_{i=1(2)}^{\infty}\frac{4h}{n\pi}\sin n\omega(t-t1) \tag{6.4.25}$$

这里 (2) 表示步距为 2 。过程输出的 Fourier 级数展开可以写成

$$c(t)=\sum_{i=1(2)}^{\infty}\frac{4h}{n\pi}g_n\sin[n\omega(t-t_1)+\phi_n] \tag{6.4.26}$$

且 g_n 和 ϕ_n 分别为受控对象的幅值和相位，即 $G(jn\omega)=g_n e^{j\phi_n}$ 。若系统的外部输入为 0，则有 $x(t)=-c(t)$，且切换点处满足 $x(t_1)=\delta, \dot{x}(t_1)<0$ 。这样就可以定义一个 $A(\omega)$ 轨迹

$$\Re e[A_G(\theta,\omega)]=\sum_{i=1(2)}^{\infty}\left[V_G(n\theta)\sin(n\theta)+U_G(n\theta)\cos(n\theta)\right] \tag{6.4.27}$$

$$\Im m[A_G(\theta,\omega)]=\sum_{i=1(2)}^{\infty}\left[\frac{1}{n}V_G(n\theta)\cos(n\theta)-U_G(n\theta)\sin(n\theta)\right] \tag{6.4.28}$$

这里 $G(jn\omega)=U_G(n\omega)+jV_G(n\omega)$ 。假设 $t_1=0$，极限环的幅值和频率可由下式解出

$$\Im m[A_G(0,\omega)+A_G(\omega\Delta t,\omega)]=-\frac{\pi\delta}{2h} \tag{6.4.29}$$

并满足约束条件 $\Re e[A_G(0,\omega)-A_G(\omega\Delta t,\omega)]<0$，若继电特性对称则可得出

$$\Im m[A_G(0,\omega)]=-\frac{\pi\delta}{4h} \tag{6.4.30}$$

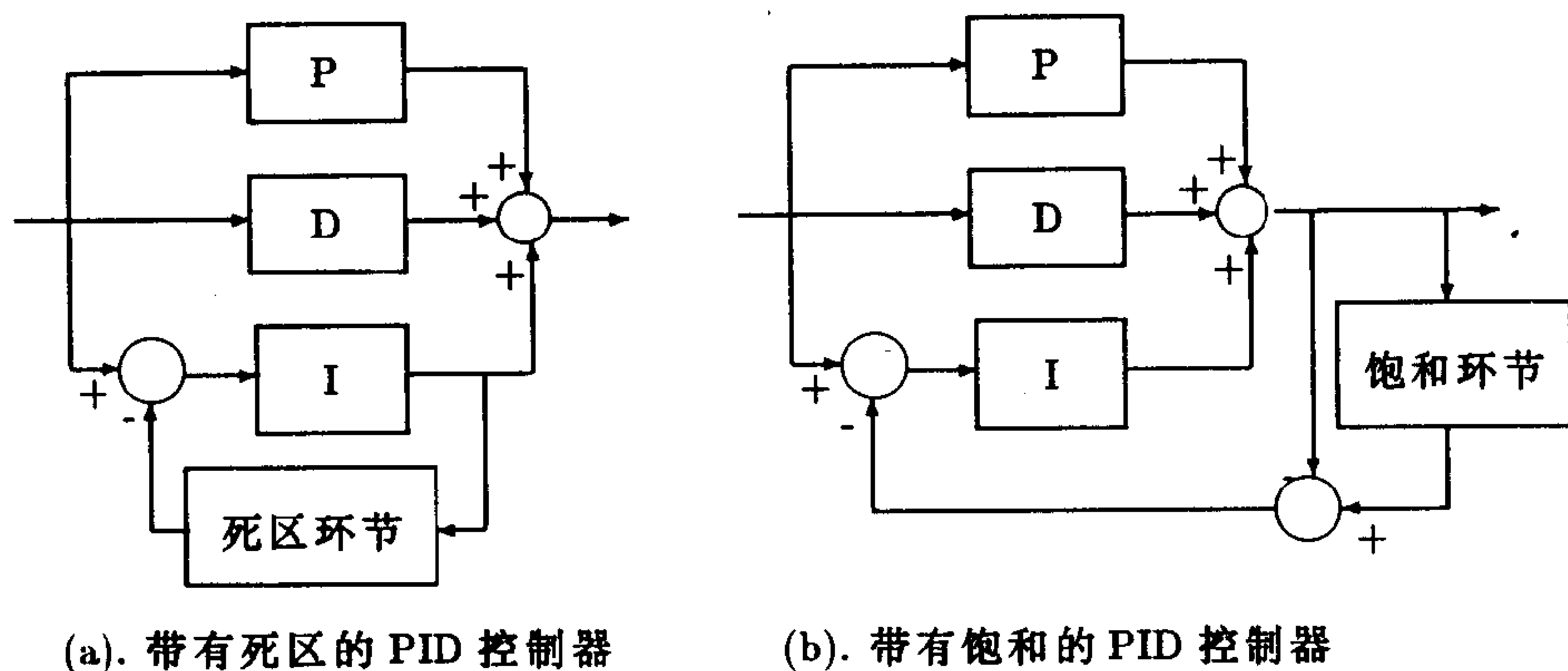
- **仿真分析法**: 文献 [36] 中给出了仿真方法的建议，其实也可以采用第 5 章中给出的非线性系统频率响应方法来求取 ω_c 和 K_c 。

典型 PID 控制也有其弊病，例如对 PID 控制有下面两个公共的认识：

- **启动回绕现象** (reset windup): 在启动过程中，因为误差信号会长时间保持较大的值，所以控制器的积分部分的输出将很大，它将导致控制信号趋于极限值，这样当过程输出发生变化时，控制器输出仍然处于极限状态，而积分器仍然发生积分动作，这样的饱和现象称为启动回绕现象。在实际应用中往往需要对标准 PID 控制的结构加以改善，例如引入非线性 PID 控制结构来避免这样的现象。
- **微分突变现象** (derivative kick): 由于 PID 控制中存在微分或近于微分的动作，所以对进入微分环节的信号若出现跃变的现象，则控制器输出将出现一个尖峰，称作微分突变，这也是需要在实际中避免的问题，如果将微分动作放在输出信号上，因为输出信号一般较平稳，所以往往能较好地抑制微分突变的现象。

为了解决上面提出的问题，还可以在 PID 控制器的一些部分加上非线性环节，例如可以构造出如图 6-35 (a), (b) 所示的非线性 PID 结构，其中前一种结构中当积分器的输入信号达到一定的范围时死区非线性环节将起作用，这将削弱积分器的作用，从而可以在一定程度上抑制启动回绕。在第 2 种非线性结构下，当控制器的输出超出了一定的限度，则积分器的输入信号将被减弱，从而使得积分器的作用被减弱。

文献 [3] 中仔细分析了前面定义的规范化参数 τ 和 κ，并对它们不同的取值建议了校正策略，如表 6-9 所示。在这里除了前面定义的 τ 和 κ 常数之外，还定义了 τ_2 和 κ_2



(a). 带有死区的 PID 控制器　　(b). 带有饱和的 PID 控制器

图 6-35　非线性 PID 控制结构

数值，它们是针对含有积分的对象模型 $G(s) = K_v e^{-sL}/[s(1+sT_v)]$ 而定义的规范化系数，具体的定义及其间关系为

$$\tau_2 = \frac{L}{T_v},\ \kappa_2 = \frac{\lim\limits_{s\to 0} sG(s)}{\omega_c|G(j\omega_c)|} = \frac{1}{2\pi}K_v K_c T_c,\ 且\ \tau_2 = \frac{\frac{2}{\pi} + \operatorname{atan}\sqrt{\kappa_2^2-1}}{\sqrt{\kappa_2^2-1}} \tag{6.4.31}$$

表 6-9　时间延迟对象控制器选择表

τ 或 κ 范围	不需精密控制	需要精密控制		
		高噪声	低饱和限	低量测噪声高饱和限
$\tau > 1, \kappa < 1.5$	I 控制	I+B+C	PI+B+C	PI+B+C
$0.6 < \tau < 1, 1.5 < \kappa < 2.25$	I 或 PI 控制	I+A	PI+A	PI+A+C 或 PID+A+C
$0.15 < \tau < 0.6, 2.25 < \kappa < 15$	PI 控制	PI	PI 或 PID	PID
$\tau < 0.15, \kappa > 15$ 或 $\tau_2 > 0.3, \kappa_2 < 2$	P 或 PI 控制	PI	PI 或 PID	PI 或 PID
$\tau_2 < 0.3, \kappa_2 > 2$	PD+E	F	PD+E	PD+E

A 表示建议前馈补偿，B 表示需要前馈补偿，C 表示建议死区补偿，D 表示需要死区补偿，E 表示需要设定点加权，F 表示极点配置。

文献 [12] 中还探讨了对 PID 控制的 3 个信号作非线性处理而构成的非线性 PID 结构，当然用户可以利用 MATLAB 这样的方便工具对这种控制结构做进一步的研究。

6.4.3 伪微分反馈控制方案

在控制系统的设计中有一类控制器称为伪微分反馈控制器 (pseudo-derivative feedback, 简称 PDF)，最早是由 Phalen 给出的 [28]，该控制策略如图 6-36 所示，在

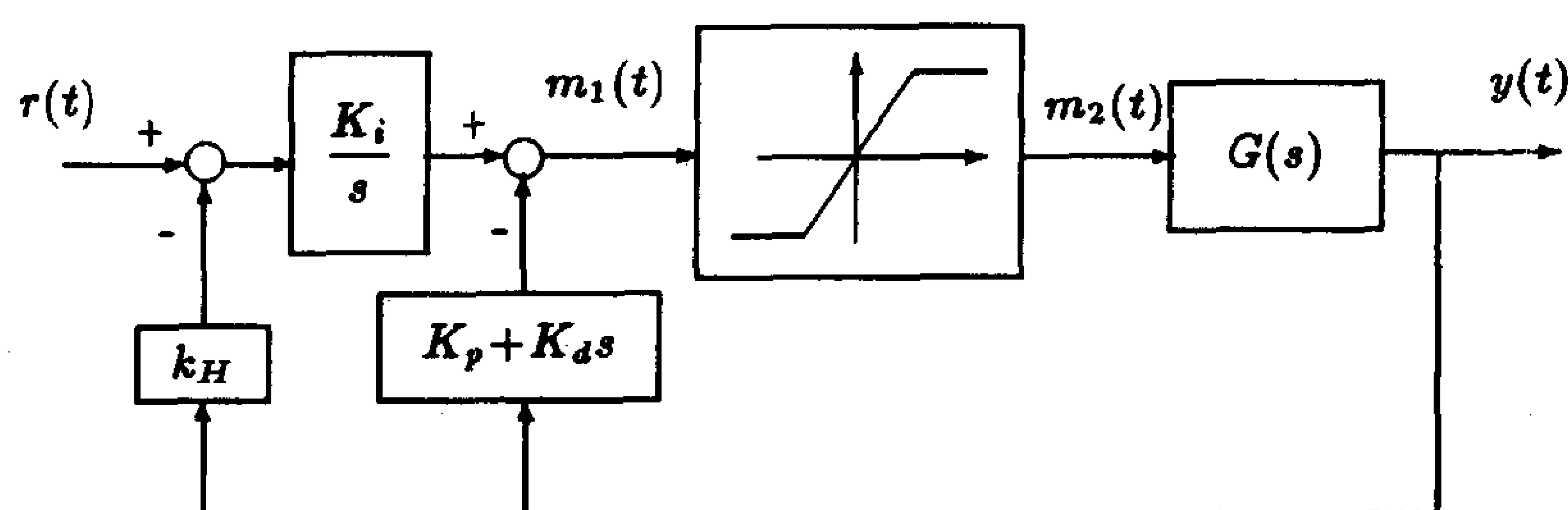


图6-36 伪微分反馈控制策略

该算法中反馈环节为 $K_D(s)=K_ds+K_p$, 并引入了积分环节 K_i/s 来确保系统的闭环响应中不存在静态误差，系统的外反馈回路的 k_H 用来调节系统输出的幅值。为了保证进入对象环节的信号 $m_2(t)$ 不至于过大，在受控对象前还加入了一个饱和限幅的非线性环节。可以证明[28]这样的控制器有着比传统的 PID 控制器更明显的优越性，因为它的前向通路是由单一的积分环节驱动的，而微分环节是对较为平滑的输出信号作用的，所以并不存在像传统 PID 控制中微分突变的现象，所以有着较好的控制效果。

在控制器设计算法的推导中，假设信号 $m_1(t)$ 足够小，使得该信号在后面的饱和非线性环节中的线性段内工作，这样就可以在推导中忽略后面的非线性环节，从而将系统等效成一般的线性系统，这样可以大大地简化控制器设计算法的运算量。可以由后面给出的例子看出，尽管做了这样的假设，设计出来的 PDF 控制器的行为还是很令人满意的，即使存在很大的控制器参数扰动仍然如此。

假设受控对象环节 $G(s)$ 为一个 2 阶线性传递函数

$$G(s)=\frac{b}{s^2+a_1s+a_2} \tag{6.4.32}$$

则内环路的闭环传递函数可以写成

$$G_I(s)=\frac{b}{s^2+(a_1+bK_d)s+(a_2+bK_p)}=\frac{b}{s^2+2\zeta\omega_ns+\omega_n^2} \tag{6.4.33}$$

这里记 $2\zeta\omega_n=a_1+bK_d$, 且 $\omega_n^2=a_2+bK_p$, 这样就可以证明整个系统的闭环传递函数为

$$G_{cl}(s)=\frac{bK_i}{s^3+2\zeta\omega_ns^2+\omega_n^2s+bk_HK_i} \tag{6.4.34}$$

将上面多项式写成

$$s^3+2\zeta\omega_ns^2+\omega_n^2s+bk_HK_i=(s+\alpha)\Big[s^2+(2\zeta\omega_n-\alpha)s+\omega_n^2-\alpha(2\zeta\omega_n-\alpha)\Big] \tag{6.4.35}$$

其中 $bk_HK_i=\alpha\Big[\omega_n^2-\alpha(2\zeta\omega_n-\alpha)\Big]=\alpha^3-2\zeta\omega_n\alpha^2+\omega_n^2\alpha=\mathcal{F}(\alpha)$, 对上述的模型如果想保证所有的闭环极点都为负实数，则从式 (6.4.35) 可以容易地得出

$$\Delta=(2\zeta\omega_n-\alpha)^2-4\Big[\omega_n^2-\alpha(2\zeta\omega_n-\alpha)\Big]\geq 0 \tag{6.4.36}$$

记 $\alpha_{\min} \le \alpha \le \alpha_{\max}$, 其中

$$\alpha_{\min} = \frac{2\zeta - 2\sqrt{4\zeta^2 - 3}}{3}\omega_n, \quad \alpha_{\max} = \frac{2\zeta + 2\sqrt{4\zeta^2 - 3}}{3}\omega_n \tag{6.4.37}$$

且 $4\zeta^2 - 3 \ge 0$。为了获得最快速的闭环响应，可以在允许的条件下将 K_i 的值尽可能取成大值，从而得出下面的式子

$$\frac{d\mathcal{F}(\alpha)}{d\alpha} = 3\alpha^2 - 4\zeta\omega_n\alpha + \omega_n^2 = 0, \quad \frac{d^2\mathcal{F}(\alpha)}{d\alpha^2} = 6\alpha - 4\zeta\omega_n < 0 \tag{6.4.38}$$

由上面的第一个式子将得出 $\alpha^* = \left(2\zeta - \sqrt{4\zeta^2 - 3}\right)\omega_n/3$, 很显然 $\alpha^* \in [\alpha_{\min}, \alpha_{\max}]$, 这样可以容易地看出

$$K_i^* = \frac{\mathcal{F}(\alpha^*)}{bk_H} = \frac{\omega_n^3}{27bk_H}\left[2\sqrt{(4\zeta^2 - 3)^3} - 16\zeta^3 + 18\zeta\right] \tag{6.4.39}$$

对 K_i^* 求导可以得出

$$\frac{dK_i^*}{d\zeta} = -\frac{2\omega_n^3}{9bk_H}\left(2\zeta - \sqrt{4\zeta^2 - 3}\right)^2 \le 0 \tag{6.4.40}$$

可见，为了使 K_i^* 取最大值，ζ 应该尽可能取小值，亦即

$$\zeta = \sqrt{3}/4 = 0.866, \quad 且 \ K_i^* = \frac{\sqrt{3}}{9}\frac{\omega_n^3}{bk_H} \tag{6.4.41}$$

可以看出 $\alpha_{\min} = \alpha_{\max} = \sqrt{3}\omega_n/3$, 故

$$\alpha^* = \frac{\sqrt{3}}{3}\omega_n, \quad 且 \ K_i^* = \frac{\alpha^{*3}}{bk_H} \tag{6.4.42}$$

进一步地可见其它两个闭环极点也等于 $-\alpha^*$, 这时闭环系统的输出信号的 Laplace 变换 $Y(s)$ 可以由下式求出

$$Y(s) = \frac{1}{k_H}\frac{\alpha^{*3}}{s(s+\alpha^*)^3} = \frac{1}{k_H s} + \frac{\beta_1}{(s+\alpha^*)^3} + \frac{\beta_2}{(s+\alpha^*)^2} + \frac{\beta_3}{(s+\alpha^*)} \tag{6.4.43}$$

从中可以得出 $\beta_1 = -\alpha^{*2}/k_H$, $\beta_2 = -\alpha^*/k_H$, $\beta_3 = -1/k_H$, 且控制系统的输出 $y(t)$ 为

$$y(t) = \frac{1}{k_H}\left[1 - \left(\frac{1}{2}\alpha^{*2}t^2 + \alpha^* t + 1\right)\mathrm{e}^{-\alpha^* t}\right] = \frac{1}{k_H}\left[1 - \left(\frac{1}{2}\tau^2 + \tau + 1\right)\mathrm{e}^{-\tau}\right] \tag{6.4.44}$$

式中 $\tau = \alpha^* t$. 如果期望系统具有调节时间 t_s，则要求有 $y(t_s) = 0.98/k_H$, 这样就可以得出非线性方程 $(0.5\tau^2 + \tau + 1)\mathrm{e}^{-\tau} = 0.02$, 求解该方程可以得出 $\tau = 7.5167$, 从而可以得出

$$\alpha^* = 7.5167/t_s \tag{6.4.45}$$

由上面的推导很自然就可以得出系统的 PDF 控制策略的控制器参数为

$$K_d = \frac{1}{b}\left(3\alpha^* - a_1\right), \quad K_p = \frac{1}{b}\left(3\alpha^{*2} - a_2\right), \quad K_i = \frac{\alpha^{*3}}{bk_H} \tag{6.4.46}$$

模型降阶技术在最近的 30 年中是很引人注目的，并取得了很大的进展，为使得降阶模型的响应更逼近于原系统，可以直接使用文献 [31] 中给出的最优降阶算法程序。如果原系统为高阶的模型，则可以利用模型降阶技术来获得一个分子为 0 阶、分母为 2 阶的降阶模型，当然可以使用各种降阶方法，如 Padé 降阶算法，主导模态算法及最优降阶算法等，然后根据得出的降阶模型，套用前面的 PDF 控制器设计算法，就可以得出针对原高阶模型的 PDF 控制器来。求取降阶模型是一件很容易的事，例如 Padé 和主导模态降阶算法可以通过计算器来简单地实现，从而获得合适的降阶模型。

例 6.11　这里将给出一个典型的系统实例来演示 PDF 控制器设计的算法，并在控制器参数及受控对象参数变化时对原系统作仿真分析，来观测系统的闭环响应效果。

考虑下面给出的一个 4 阶模型

$$G(s) = \frac{6s^3 + 26s^2 + 6s + 20}{s^4 + 3s^3 + 4s^2 + 2s + 2}$$

并假定控制系统中非线性饱和限幅环节的上下限分别为 $u_m = 2.5$ 和 $l_m = -2.5$ 。使用 Padé 近似和主导模态方法可以分别获得系统的 2 阶近似模型为

$$G^p_{0/2}(s) = \frac{20.4082}{s^2 + 1.4286s + 2.0408}, \quad G^d_{0/2}(s) = \frac{6.2057}{s^2 + 0.0531s + 0.6206}$$

此外可以获得一个使得原模型和降阶模型之间误差平方的积分为最小的最优降阶模型为

$$G^*_{0/2}(s) = \frac{39.0637}{s^2 + 3.7052s + 3.9064}$$

原系统与各个降阶模型的开环阶跃响应曲线如图 6-37 所示，可以看出这 3 个降阶模型的开环阶跃响应和原系统并不是特别相象。

表6-10　各种情况下 PDF 控制器的参数表

t_s	降阶模型	K_i	K_p	K_d
1.5	$G^p_{0/2}(s)$	7.584	4.148	0.7193
	$G^d_{0/2}(s)$	24.94	13.84	2.587
	$G^*_{0/2}(s)$	3.2213	1.8285	0.2900
0.5	$G^p_{0/2}(s)$	166.5	33.12	2.14
	$G^d_{0/2}(s)$	547.5	109.2	7.259
	$G^*_{0/2}(s)$	86.9757	17.2565	1.0597
0.1	$G^p_{0/2}(s)$	20810	830.5	10.98
	$G^d_{0/2}(s)$	68440	2731	36.33
	$G^*_{0/2}(s)$	10872	433.81	5.6778

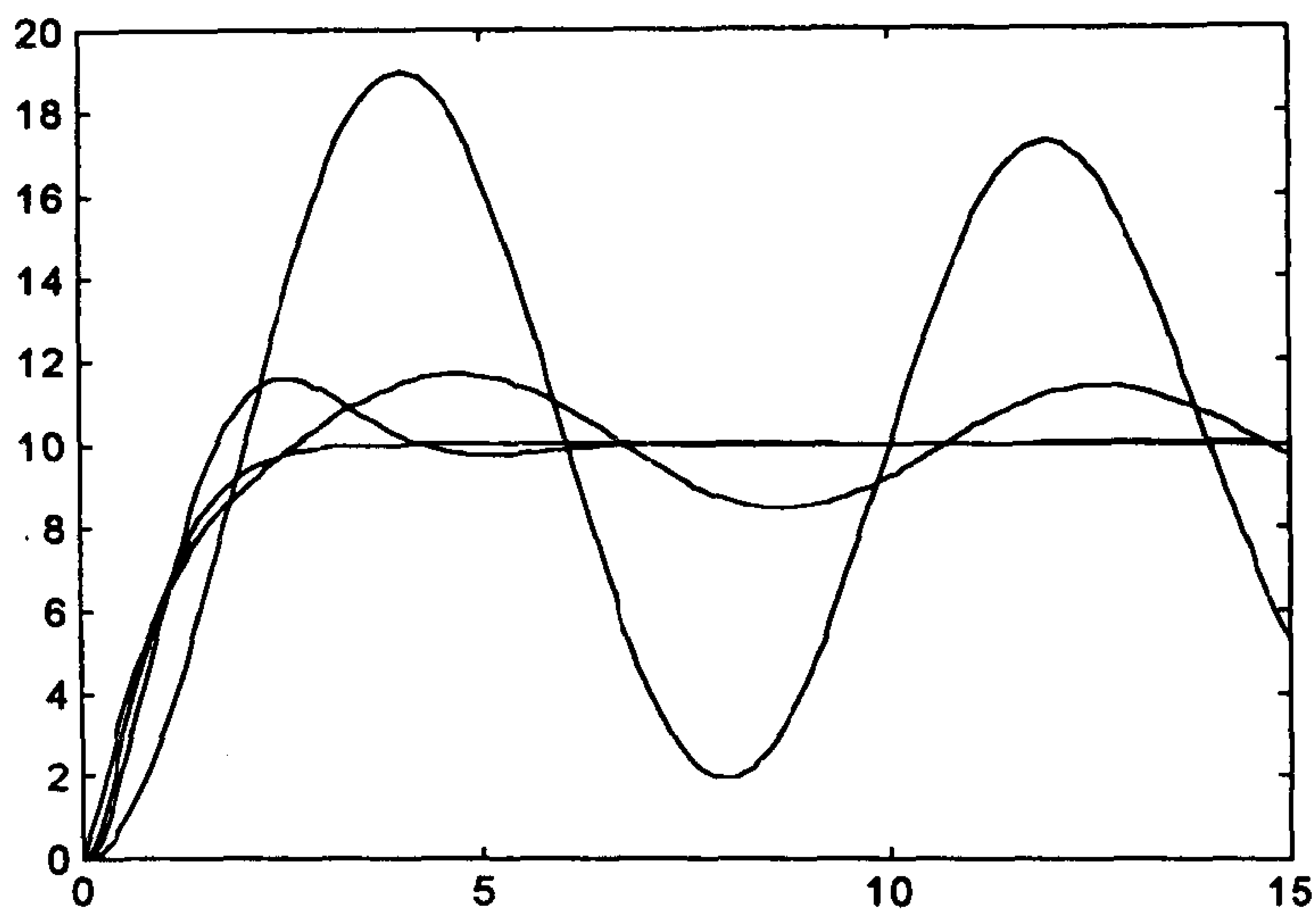


图6-37　系统模型的开环阶跃响应

这里将根据这样的降阶模型来设计出合适的 PDF 控制器来。分别给出 3 个要求的调节时间 $t_s = 1.5, 0.5, 0.1$，则使用得出的 3 个降阶模型 $G^p_{0/2}(s)$, $G^d_{0/2}(s)$, 和 $G^*_{0/2}(s)$ 就可以设计出 9 个不同的 PDF 控制器，这些控制器的参数如表 6-10 所示。对调节时间 $t_s = 0.1$ 的闭环阶跃响应曲线如图 6-38 所示，可以看出对具有较大调节时间的原系统来说，在像 0.1 秒这样苛刻的调节时间要求下，系统

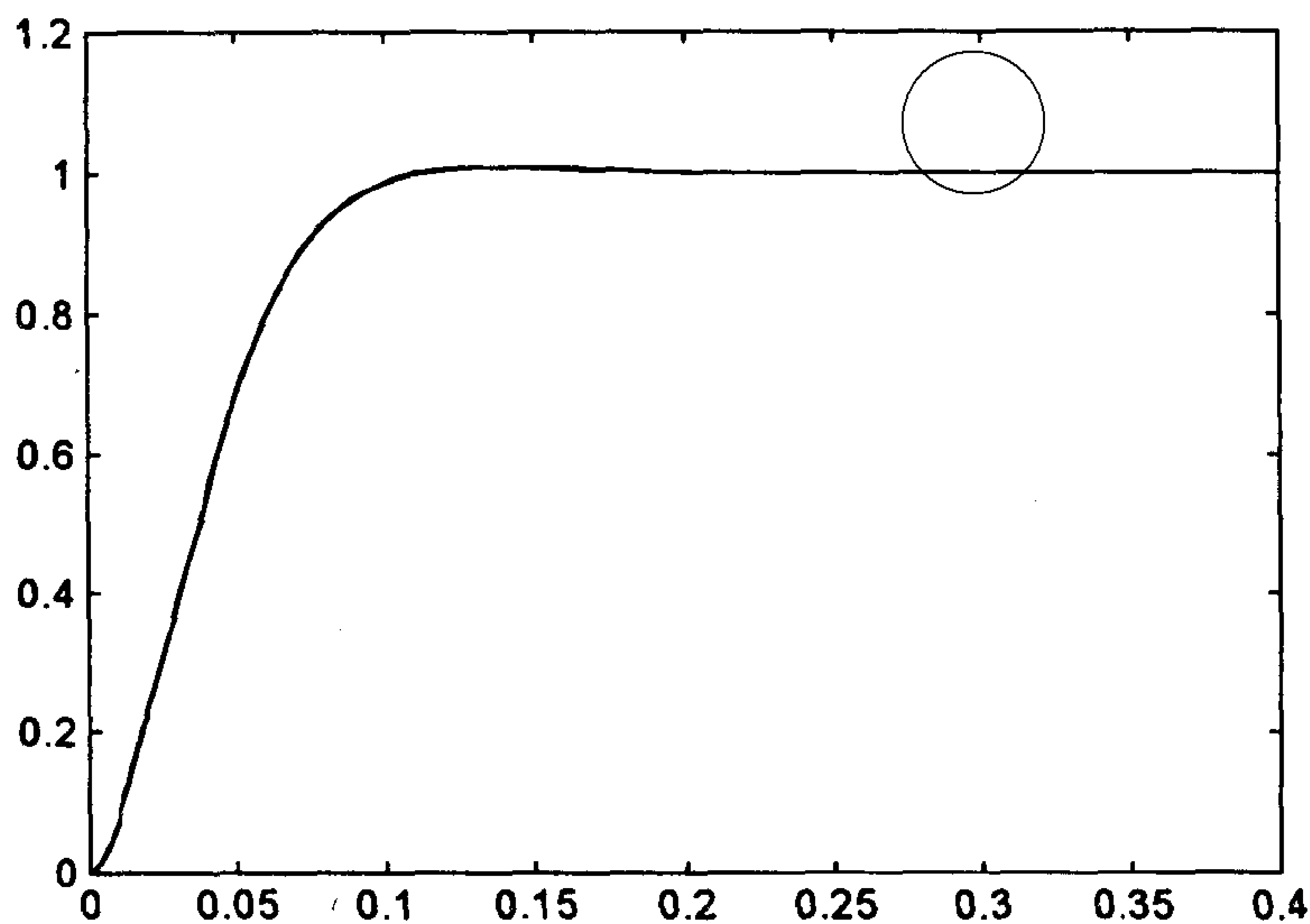


图6-38　系统模型的闭环阶跃响应

的响应还是很令人满意的，这是在其它控制策略下是很不容易做到的。同时还可以看出，尽管各个控制器的参数截然不同，但从闭环阶跃响应的曲线上难以区分。从给出的图形中还可以看出，响应初始段闭环响应并不是很平滑的，这时因为由非线性饱和限幅环节在起作用，该环节的作用就是尽量减小进入受控对象的信号幅值，在这 3 个 PDF 控制器下进入受控对象 $G(s)$ 的信号 $m_2(T)$ 的阶跃响应曲线如图 6-39 所示。

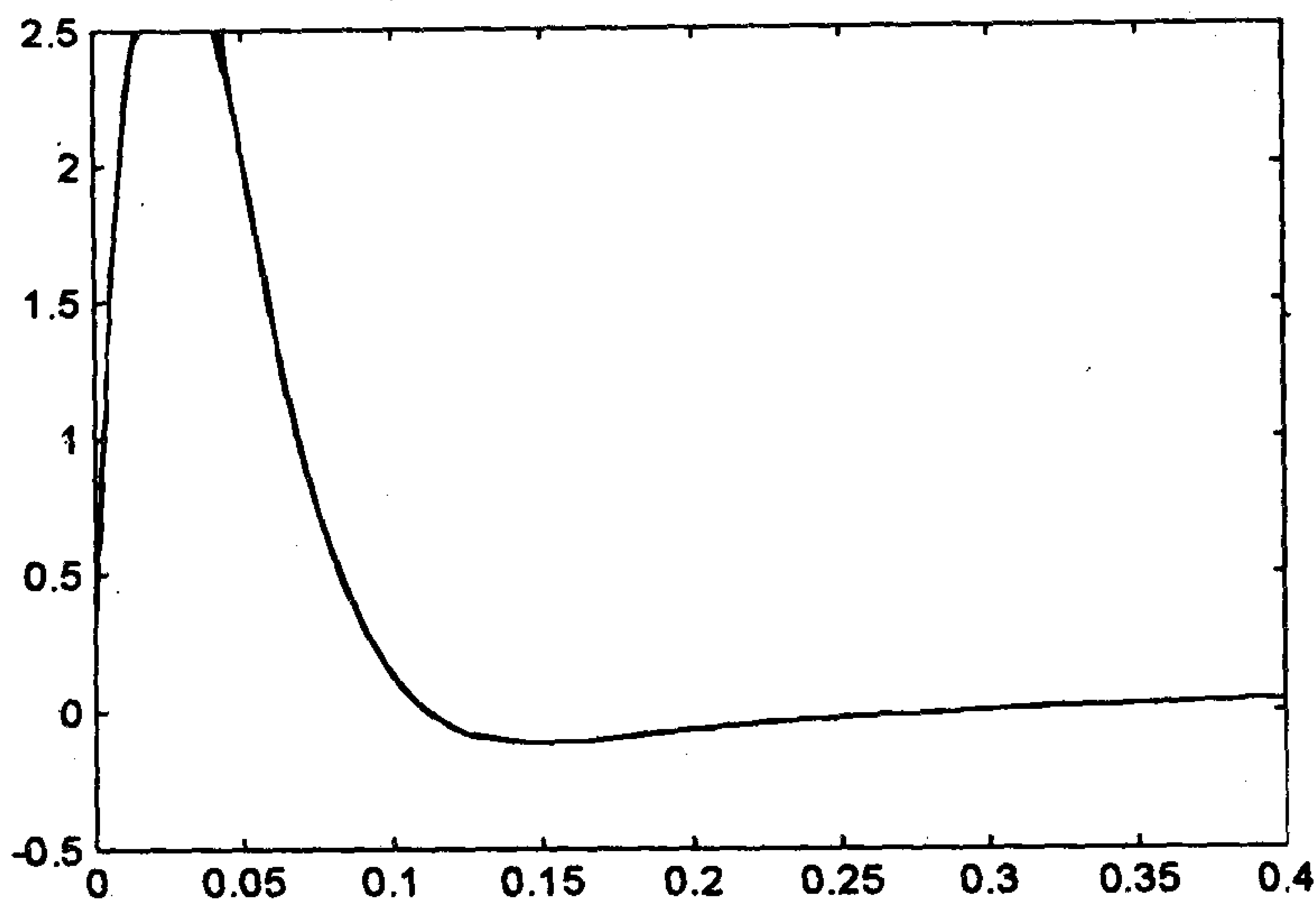


图6-39　进入受控对象的信号 $m_2(t)$

可以看出，由于PDF控制器设计算法的限制，在要求的调节时间取得很小时，往往会得出较大的 K_i 参数，这是因为 $K_i \propto 1/t_s^3$，而该参数的大小还在某种程度上取决于降阶模型的形式，如在此例子中基于最优降阶模型设计出来的PDF控制器参数比基于主导模态法得出的控制器参数要明显地小，在实践中更容易实现。

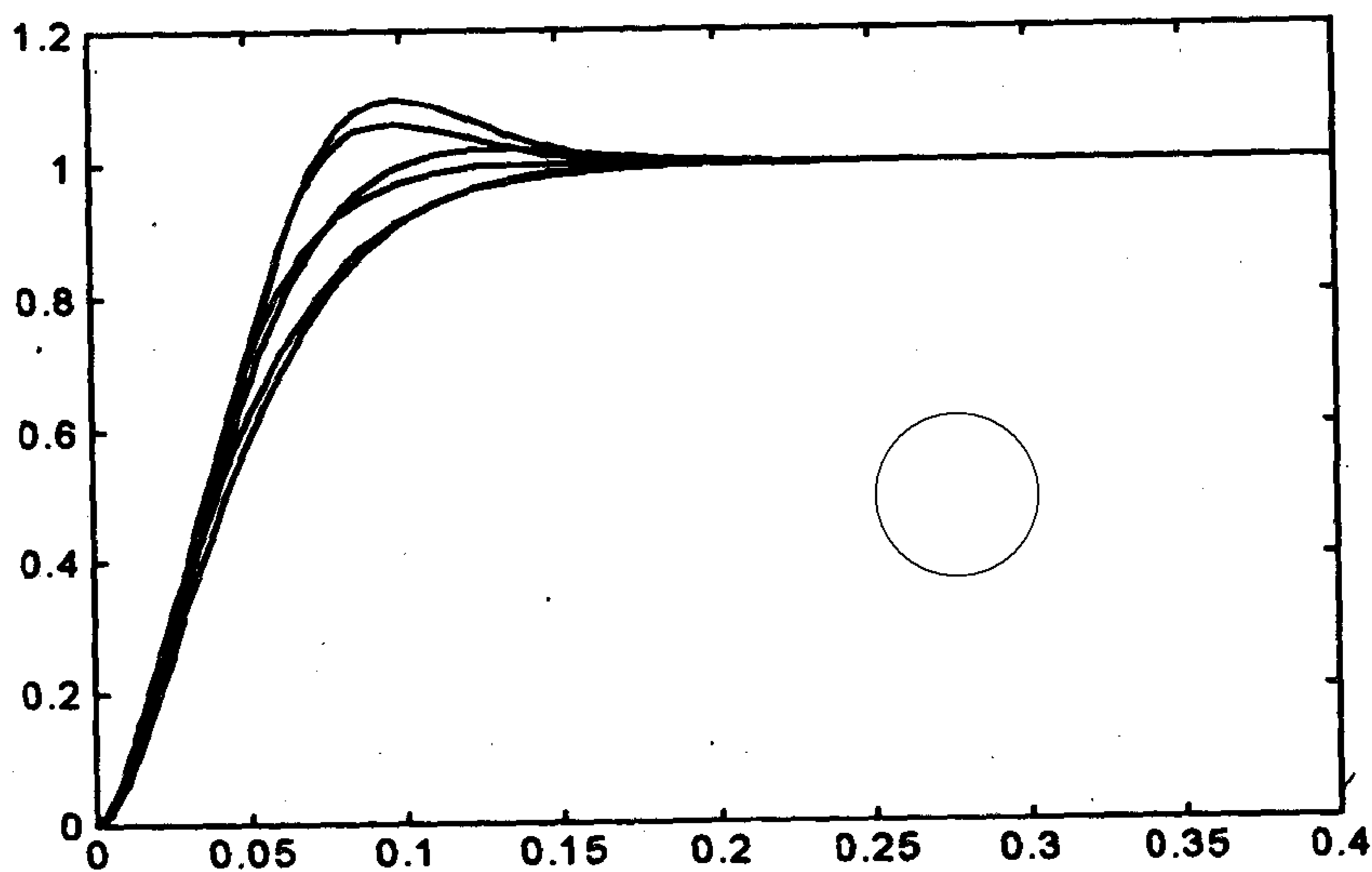


图6-40　控制器参数变化时闭环系统阶跃响应

虽然在要求快速响应时得出的 K_i 值相当大，在实际控制中实现起来较困难，但下面的分析将说明整个控制效果对控制器参数及原系统模型参数的变化并不是很明显，在 $t_s=0.1$ 下，当控制器参数 K_p，K_i 及 K_d 发生较大的变化时，闭环系统的阶跃响应 $y(t)$ 和进入对象模型的信号 $m_2(t)$ 分别如图6-40和图6-41所示。由这两个响应曲线可以容易地看出，尽管各个参数有从80%到1.2倍的变化，整个系统的闭环响应还是很令人满意的，所以即使控制器参数不准也不会对闭环控制效果有太大的影响。

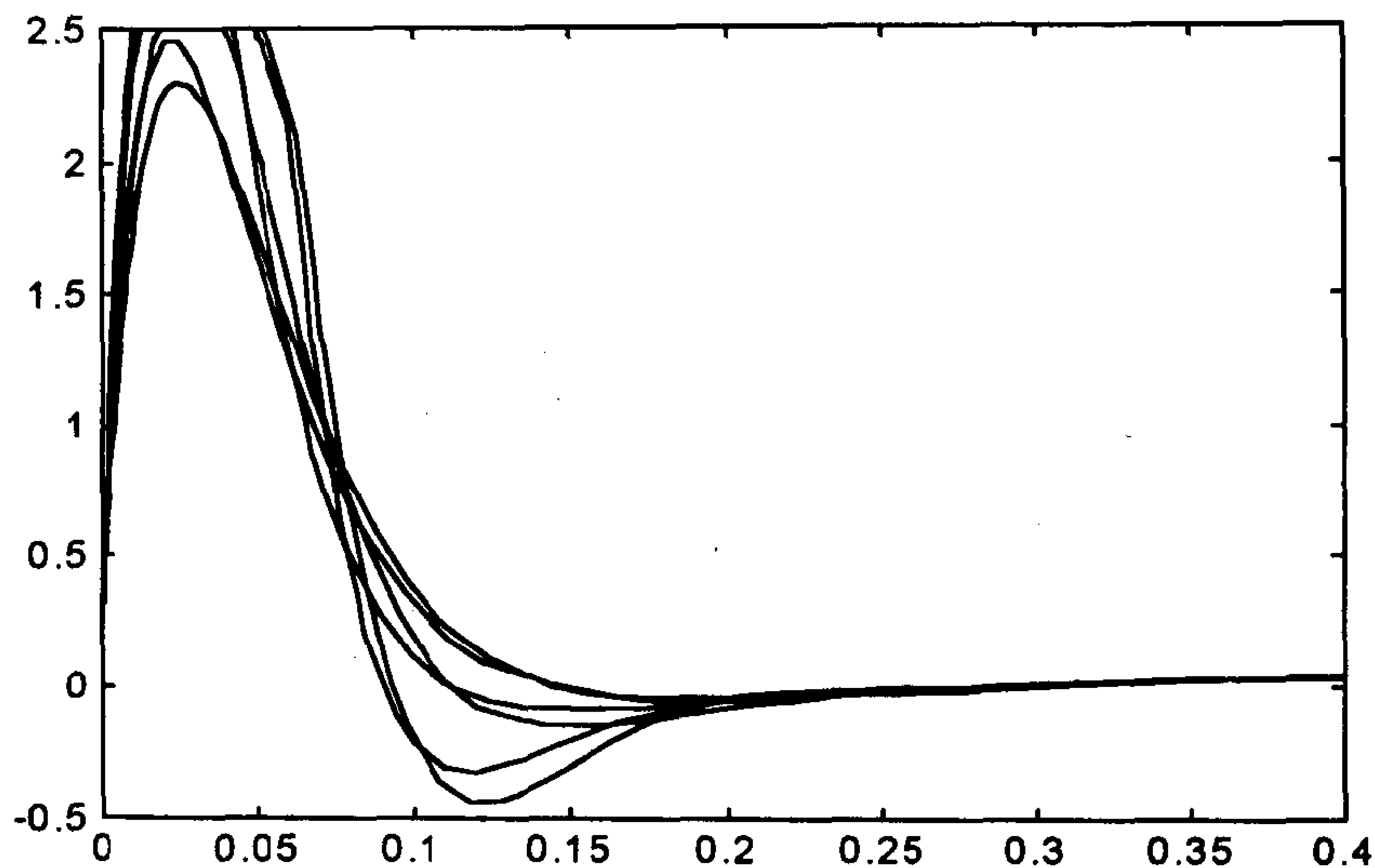


图6-41 进入对象的 $m_2(t)$ 信号

6.5 定量反馈控制设计方法

以色列学者、美国加州大学的 Issac Horowitz 教授和他的合作者们系统地提出了一种名为定量反馈理论 (quantitative feedback theory, 简称 QFT) 的设计方法 [14, 15, 16, 17]，这些方法是基于频率响应的设计方法，可以用于带有很大对象不确定性的单变量系统、多变量系统以及各种高度非线性系统和时变系统的鲁棒设计，这种设计方法既可以用于最小相位系统的设计，也可以用于非最小相位系统的设计， QFT 方法提出得比较早，但过去并未引起很高的重视，近年来，在控制界重新又对 QFT 方法发生了兴趣，并出现了 MATLAB 的 QFT 设计工具箱。虽然 QFT 方法可以用于多变量系统的设计，但该设计方法的核心是单变量最小相位系统的设计。在本节中将给出 QFT 的基本设计方法与工具箱概述。

6.5.1 单变量系统的 QFT 设计方法

由于单变量最小相位系统是 QFT 设计问题的核心，所以在这里将详细叙述单变量系统的 QFT 设计方法和思想，单变量系统的 QFT 设计步骤为：

- **设定控制系统结构**: QFT 设计下的控制系统结构如图 6-42 所示，在控制系统中 $G(s)$ 为受控对象的传递函数，它可以带有较大的不确定性，且不确定性的范围是已知的，在 QFT 设计中经常记可取受控对象的集合为 $\mathcal{G}$ 。 $C(s)$ 为系统的控制器， QFT 控制的特点是该控制器是定常的，它可以控制含有大不确定性的对象，并可以对噪声干扰有满意的抑制作用。

  由于 $C(s)$ 的主要作用是保证系统能满足鲁棒稳定性的要求，其品质性能不一定很理想，这样常常需要引入前置滤波器 $F(s)$ ，其作用是动态地补偿系统的性能。

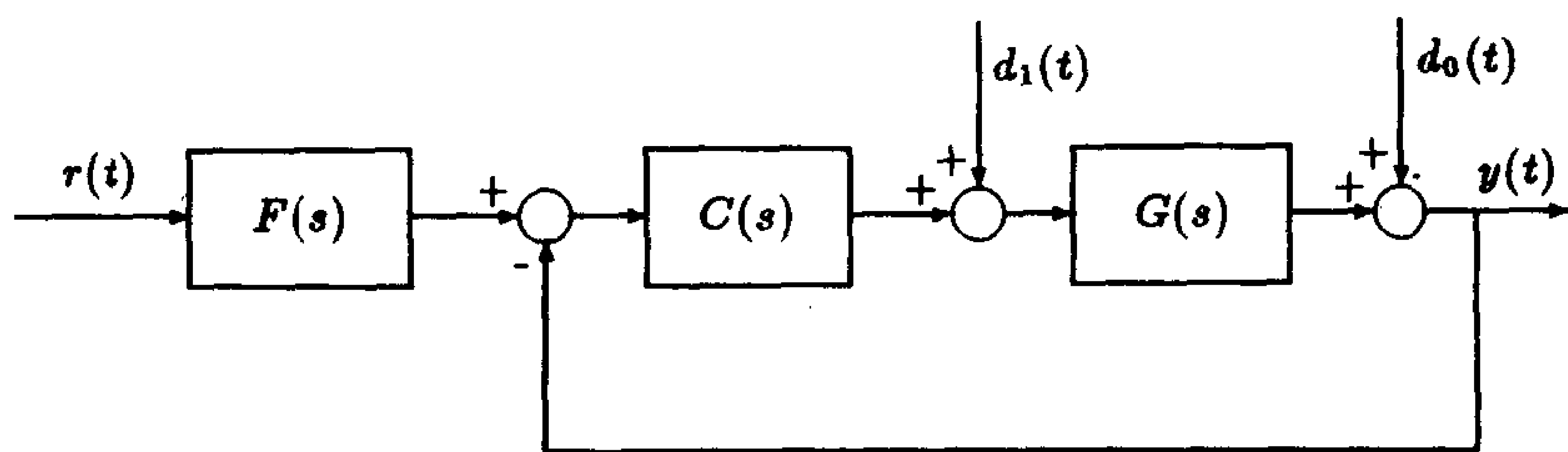


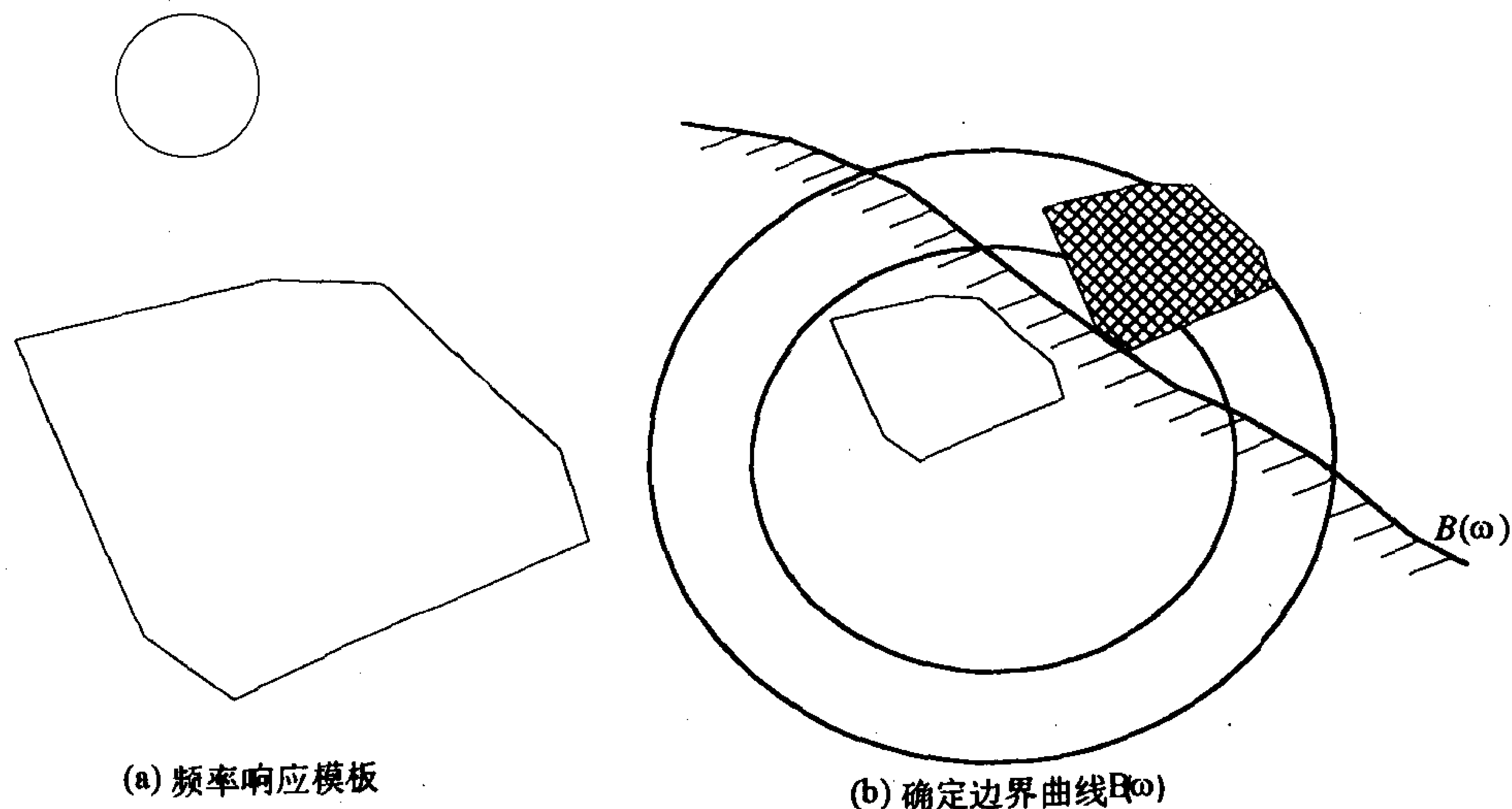
图6-42　QFT 设计的控制系统结构图

- **给定设计要求**: 在 QFT 设计中应该首先给出最终闭环系统频率响应的上下界函数 $a(\omega)$ 和 $b(\omega)$, 使得系统的闭环响应函数 $T(s)$ 满足下面的约束

$$a(\omega) \le T(j\omega) \le b(\omega) \tag{6.5.1}$$

而满足这样的频域约束即意味着满足相应的时域响应约束，关于时域和频域响应的对应关系可以参见文献 [17] 。除了频率范围的约束之外，基于稳定裕度的考虑还应该指定等 M 圆的下界 γ, 使得 $M < \gamma$ 。

- **构造频率响应模板**: 选定一个频率点 ω_1, 对此频率下的具有不确定性模型的各个选样模型进行频率响应分析，构造出 $G(s)$ 的对象模板 (plant template)，如图 6-43 (a) 所示。



(a) 频率响应模板　　(b) 确定边界曲线$B(\omega)$

图6-43　模板处理示意图

- **构造稳定边界曲线**: 在控制器 $C(s)$ 作用下，闭环系统的传递函数为

$$T(s) = [G(s)C(s)][1 + G(s)C(s)]^{-1}F(s) = L(s)[1 + L(s)]^{-1}F(s) \tag{6.5.2}$$

式中 $L(s) = G(s)C(s)$, 这样在 Nichols 图坐标系下 $L(s)$ 的模板为 $G(s)$ 对象模板的平移，其中水平的平移的幅度为 $C(s)$ 的相位角度 $\angle C(j\omega_1)$ ，而垂直方向的平移量为控

制器 $C(s)$ 的幅值 $|C(j\omega_1)|$。如图 6-43 (b) 所示。给出了上面的控制器 $C(s)$ 之后，就可以得出相应的设计要求

$$M(\omega)=\lg\left|\frac{L(j\omega)}{1+L(j\omega)}\right|\leq\gamma \tag{6.5.3}$$

其中 $\lg|L/(1+L)|$ 为等 M 圆的表示方法。

- **构造各个频率下的稳定下界**: 对各个频率 $(\omega_2,\omega_3,\cdots,\omega_m)$ 也做相应的处理，则可以得出在各个频率下的下界曲线 $B(\omega_2),B(\omega_3),\cdots,B(\omega_m)$, 这样若想得出最优的控制器，则可以由各个 $B(\omega_i)$ 曲线上选择一个点来构成 $L(j\omega)$ 曲线，但这样得出的控制器将极其复杂，所以要在 $B(\omega_i)$ 曲线允许的范围内得出"最优的"$L(s)$ 曲线，而不一定非得在边界线 $B(\omega_i)$ 上取点，由此可以设计出相应的 QFT 控制器 $C(s)$。
- **设计 $F(s)$ 控制器**: 按照前面的方法设计出来的控制器 $C(s)$ 往往不能满足式 (6.5.1) 中的频率要求，这样就需要一个前置滤波器 $F(s)$ 来使得系统满足该要求，该控制器的设计也是很简单的。由式 (6.5.2) 可以得出

$$\lg|T(j\omega)|=\lg|F(j\omega)|+\lg\left|\frac{L(j\omega)}{1+L(j\omega)}\right|=\lg|T(j\omega)|+M(\omega) \tag{6.5.4}$$

而 $\lg|T(j\omega)|$ 满足式 (6.5.1) 中的边界要求，这样 $F(s)$ 的边界可以如下确定

$$a_F(\omega)=a(\omega)-M_{\min}(\omega)\leq\lg|F(j\omega)|\leq b(\omega)-M_{\max}(\omega)=b_F(\omega) \tag{6.5.5}$$

最后根据得出的频率响应边界来设计一个前置滤波器 $F(s)$。

6.5.2 QFT 设计举例

例 6.12 本节将通过 2 阶单变量不确定系统的实例来介绍 QFT 控制系统设计的基本方法和 MATLAB 实现，并对设计过程给出必要的解释。考虑 Horowitz 给出的一个典型 2 阶不确定系统的例子 [16]

$$G(s)=\frac{k}{As^2+Bs+C}$$

假设其参数的不确定范围为：$k\in[1,4],A\in[1,4],B\in[-2,2],C\in[1,6.25]$, 注意这些不确定参数的变化范围和其本身比较起来还是很大的，且由于 B 参数可以取负值，整个受控对象可以为不稳定的，所以用单一控制器来控制并想保持较好的控制效果，采用传统的方法并不是很简单的， QFT 控制的优点就在于此。要设计出较好的控制效果，则首先应该选定控制后闭环系统的频率响应上下界 $a(\omega)$ 和 $b(\omega)$, 假设可以由 MATLAB 命令来指定其上下界

```
>> w=[0,0.1,0.2,0.5,1,2,4,8,16];
>> b=[0,0.5,1,1.5,2,1.5,-0.7,-8,-20];
>> a=[0,-0.5,-1,-2.5,-5,-10,-20,-38,-70];
>> subplot(121); semilogx(w,b,w,a,'--'); axis([.1,16,-70,2])
>> subplot(122); plot(w,b-a); axis([0,8,0,30])
```

这时得出的闭环系统频率响应上下界曲线如图 6-44 (a) 所示，还可以绘制出上下界之差 $\delta(\omega)=b(\omega)-a(\omega)$ 曲线如图 6-44 (b) 所示。

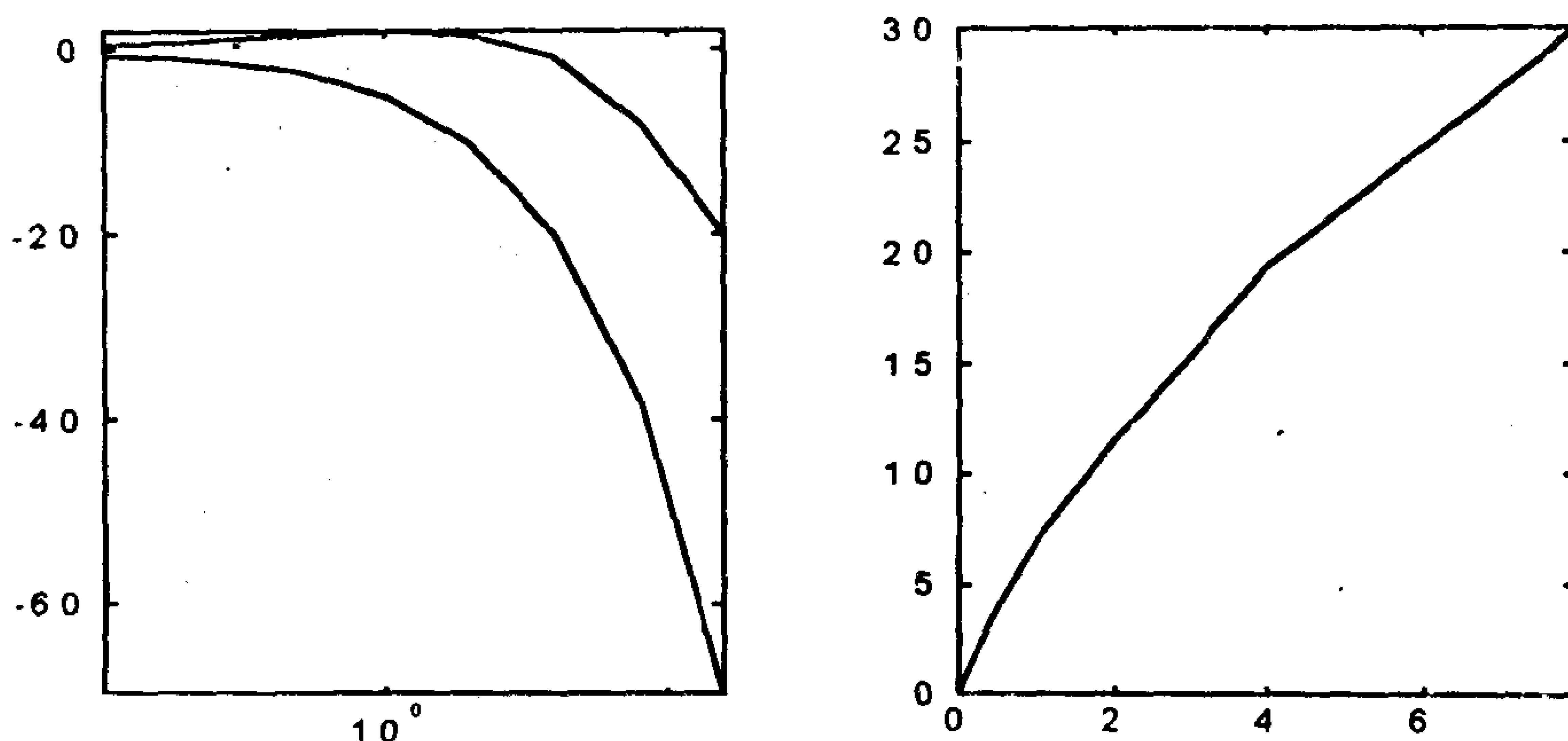


图6-44 给定的频率响应上下界

用户还可以指定 $\gamma=2.3(\text{dB})$ 来保证一定的稳定裕度，这时有

$$M(\omega)=\lg\left|\frac{L(j\omega)}{1+L(j\omega)}\right|\leq\gamma$$

由图 6-44 (b) 可以查出 $\omega=3$ 时对应的 M 的允许变化率值 $\Delta M(\omega)=\delta(\omega)$ 的值为 15.3(dB)。

要进行 QFT 设计还应选择一个标称 (nominal) 模型，在 QFT 设计方法下这个标称模型可以选作 $\mathcal{G}$ 集合中的任何一个模型，为方便起见，可以将标称模型选为上限的极端情况 $1/(4s^2+2s+6.25)$ 并把它记作 $G_N(s)$。下面将演示怎样去绘制对象模板，假设选择频率为 $\omega=3$，则对象模板可以表示为

$$\mathcal{G}(3)=\{G(j3)\}=\left\{\frac{k}{C-9A+j3B}:k\in[1,4],A\in[1,4],B\in[-2,2],C\in[1,6.25]\right\}$$

其中 k 可以首先选择为 $k=k_{\min}=1$、至于 k 的其余情况可以简单地通过将模板上移 $20\lg(4)\simeq 12\text{dB}$ 的方式获得，如果选择其中若干个点

```
>> x=[]; y=[]; x0=[]; y0=[]; z=[]; w=3;
>> for A=4:-1:1
     for B=-2:.5:2
       for C=[1:1:6 6.25]
         v=freqresp(1,[A B C],w*sqrt(-1));
         v1=angle(v); if v1>0, v1=v1-2*pi; end
         x=[x v1*180/pi]; y=[y,20*log10(abs(v))];
     end, end, end, end
>> plot(x,y,'o')
```

则可以绘制出如图 6-45 (a) 所示的模板形式，这样对 k 进行扩展 (即上移 12dB 而构成包络线)，可以给出下面 MATLAB 命令

```
>> nA=length(1:1:4); nB=length(-2:0.5:2); nC=length([1:6 6.25]);
>> ii1=[]; ii3=[];
>> for i=1:nA
     ii1=[ii1 (i-1)*nC*nB+1:(i-1)*nC*nB+nC]; ii3=[ii3 i*nC*nB-nC+1: i*nC*nB];
```

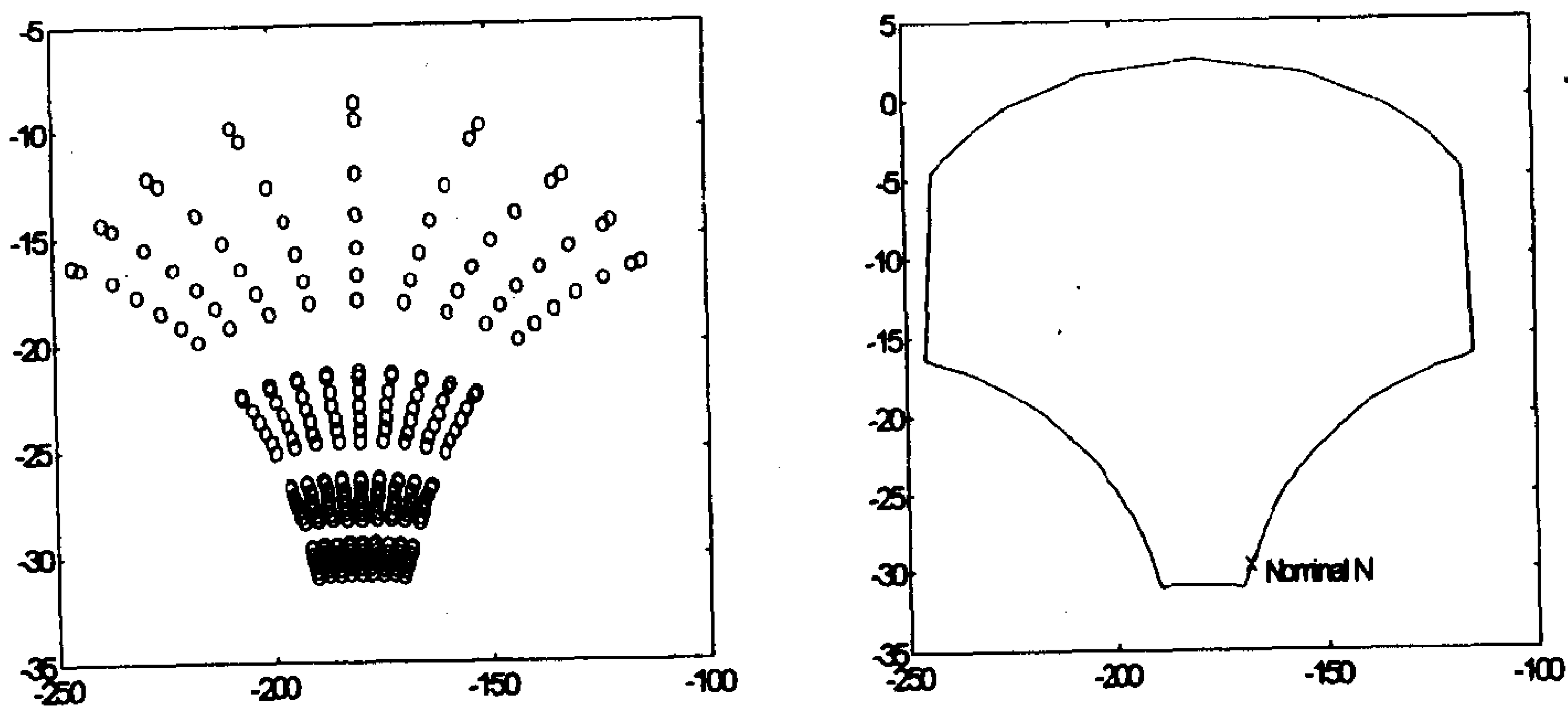


图6-45　$\omega=3$ 时对象模板示意图

```
    end
>> ii2=1:nC:nC*nB; ii4=(nA-1)*nB*nC+nC-1:nC:nA*nB*nC;
>> ii3=ii3(length(ii3):-1:1); ii2=ii2(length(ii2):-1:1);
>> x0=[x(ii1) x(ii4) x(ii3) x(ii2)]; y0=[y(ii1) y(ii4)+12 y(ii3) y(ii2)];
>> v=freqresp(1,[4,2,6.25],sqrt(-1)*3);
>> plot(x0,y0,'-', angle(v)*180/pi,20*log10(abs(v)),'x'); text(-165,-30,'Nominal N')
```

这样就可以得出如图 6-45 (b) 所示的对象模板，在对象模板上还指示出标称对象模型的频率响应点 N。

首先假设引入一个控制器 $C(s)$ 可以使得 $L(s)$ 的相位为 $-20°$，这样可以对之进行搜索，依据等 M 圆的概念总能找出模板覆盖部分的最大 M 值 $M_{\max}$ 及最小 M 值 $M_{\min}$，在垂直方向上移动模板，找出能使得 $\Delta M(\omega)=M_{\max}-M_{\min}=15.3$ 的位置，并求出 L 模板的幅值，记为 $B(\omega)$，其值为 $-10.5\angle-20°$，这时 $M_{\min}=-15.2$，$M_{\max}=0.1$，从而其差值为 15.3，在这里得出的 L 值下，控制器的值可以选择为 $19.1\angle148.6°$。对其它的相位值也进行同样的搜索，则可以得出 $B(\omega)$ 曲线，如图 6-46 所示 [16]。

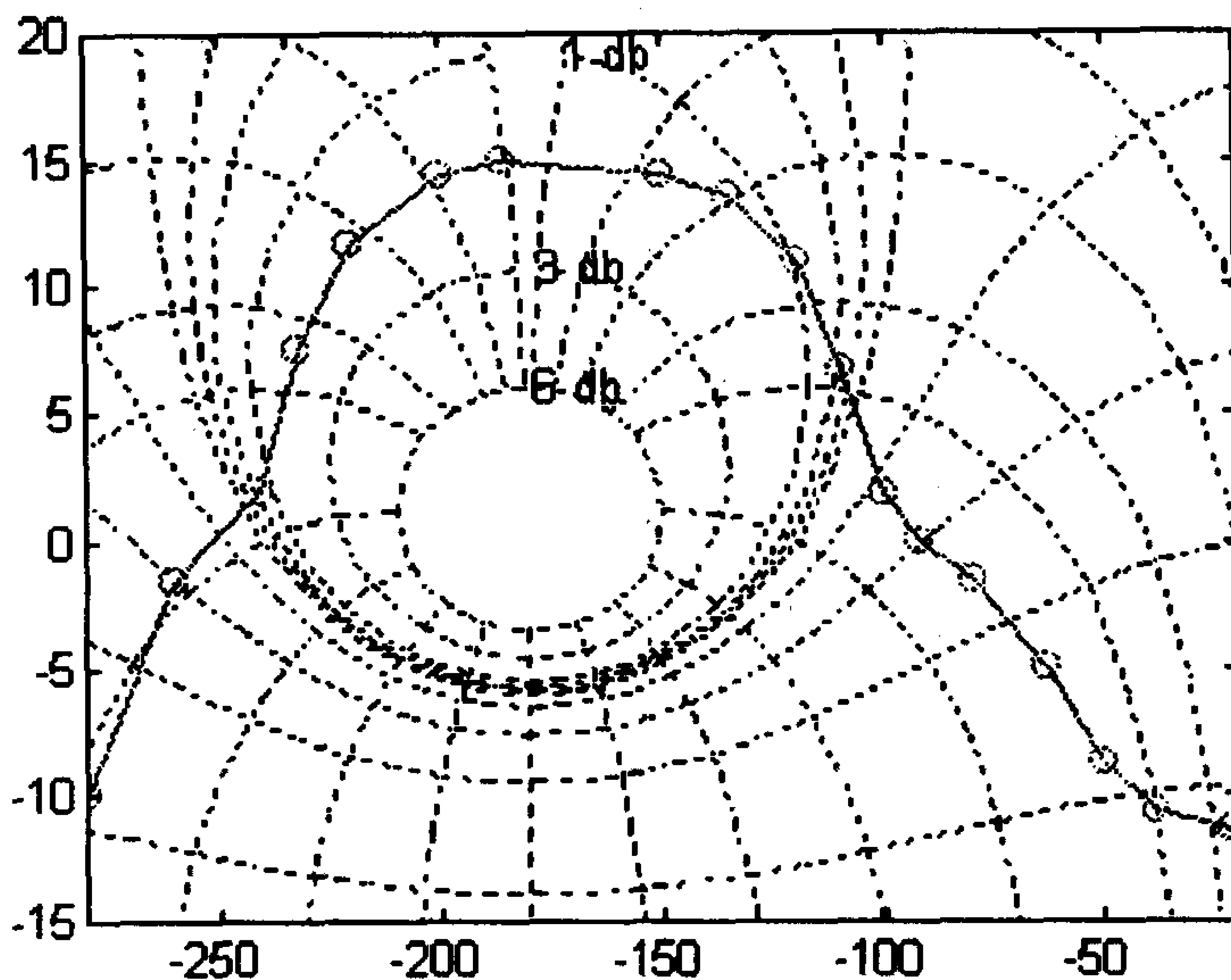


图6-46　$\omega=3$ 时稳定下限 $B(\omega)$ 图形

除了对 $\omega=3$ 选择频率获得一条 $B(\omega)$ 曲线之外，用户还可以改变频率的值、从而对其它选定的频率也绘制出响应的 $B(\omega)$ 曲线，由这种方法对一些选定的频率得出 $B(\omega)$ 曲线，而这些曲线就可以作为每个频率点处的下限边界用于系统设计了。

注意，对高频区域来说，经常可以得出一个闭合的稳定边界 $B(\omega)$，因其形状像烟灰罐 (ashcan) 而得名，这样最终系统的 $L(s)$ 高频响应不进入该区域就可以获得指定稳定裕度的效果。

文献 [16] 中给出了这样设计出来的第一个控制器，使得标称传递函数 $L(s)=9.4/s$，则在 Nichols 坐标下可以得出一条垂直的直线，它们在各个频率点处满足稳定边界 $B(\omega)$ 的要求，但这样的设计过于保守，所以还可以进一步改进 $L(s)$，例如将它设计成 $L_1(s)=9.4\times8.5/[s(s+8.4)]$，实践证明，这样的设计仍趋于保守，可以使得它的高频部分进一步衰减，从而引入标称 $L_N(s)$ 和

$$L_N(s)=\frac{9.4\times8.5}{s(s+8.4)}\left(\frac{s}{14}+1\right)\frac{280^2}{s^2+1.2\times280s+280^2}$$

这样得出的控制器 $C(s)$ 模型为

$$C(s)=\frac{L_N(s)}{G_N(s)}=\frac{9.4\times8.5\times280^2(s+14)(4s^2+2s+6.25)}{14s(s+8.5)(s^2+1.2\times280s+280^2)}$$

利用此控制器去控制不确定系统 $G(s)$，则可以得出如图 6-47(a) 所示的阶跃响应曲线，且可以得出其 Bode 频率响应，如图 6-47 (b) 所示。这些阶跃响应曲线是由下面的 MATLAB 命令得出的

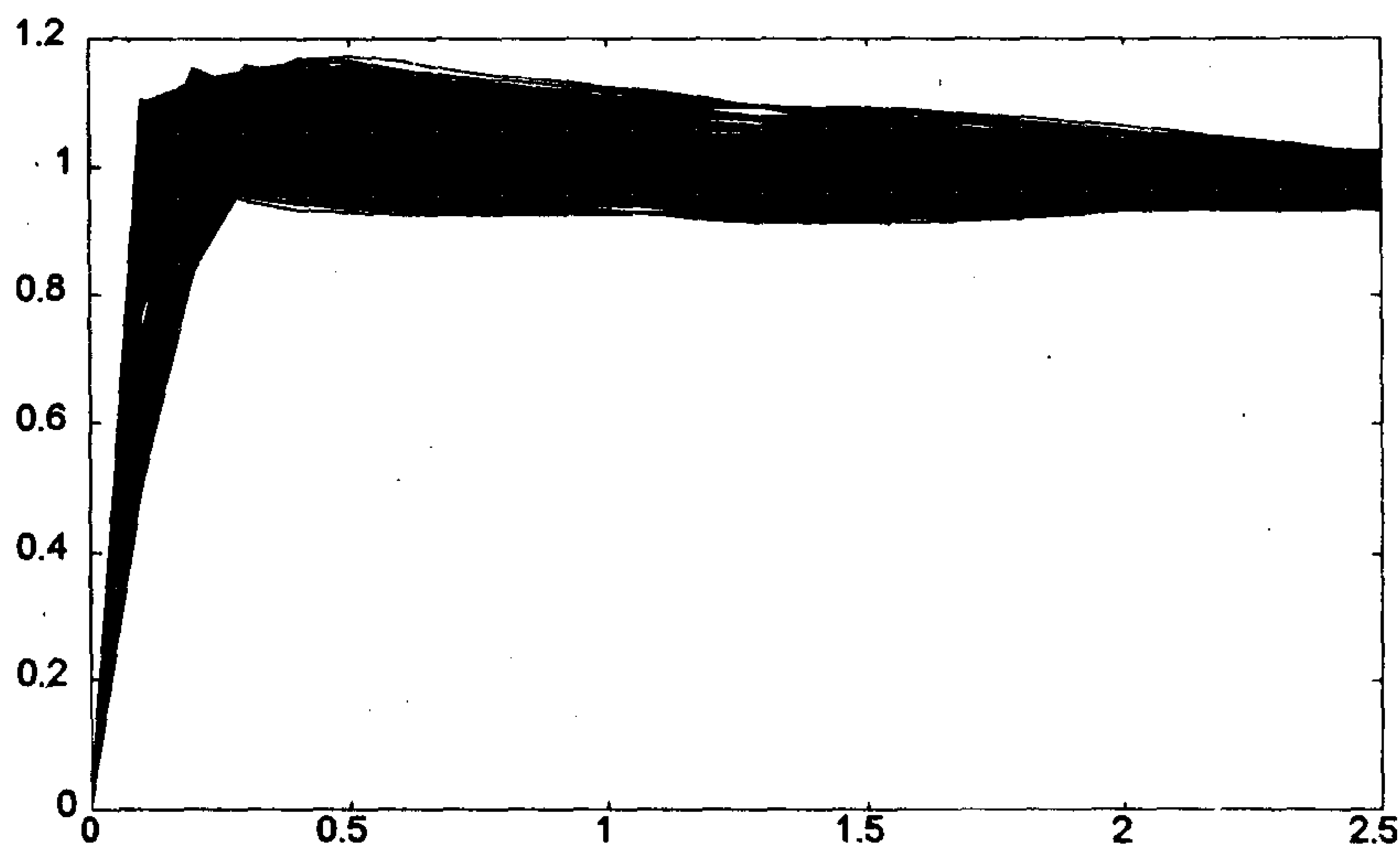


图6-47 不确定系统的闭环阶跃响应和 Bode 图

```
>> numC=9.4*8.5*280^2*conv([1,14],[4,2,6.25]);
>> denC=14*[conv([1 8.5],[1,1.2*280,280^2]) 0];
>> t=0:.1:2.5; y=[]; i=0;
>> for k=1:4
      for A=1:0.5:4
         for B=-2:0.5:2
            for C=[1:5, 6.25]
               num=numC*k; den=conv([A,B,C],denC)+[zeros(1,3) num];
               y=[y step(num,den,t)]; i=i+1;
```

```
    end, end, end, end
>> plot(t,y)
```

这里总共取了不确定模型中的 1512 个样本，从总体效果看来，这些阶跃响应曲线都是比较令人满意的，因为对象模型既可能为稳定的，也可能为不稳定的 (如 $B \le 0$)。

很容易检验这样得出的闭环系统频率响应不会满足式 (6.5.1) 中给出的边界要求，所以可以按照前面给出的方法设计出前置控制器 $F(s)$，在设计之前可以得出在选定的频率点处的 $M_{\max}$ 和 $M_{\min}$ 的值，这样可以由式 (6.5.2) 构造出表 6-11，并设计出一个满足式 (6.5.5) 频率响应要求的前置滤波器 $F(s) = 21/(s+3)(s+7)$ 来，其实从给定的数据中可以看出该滤波器更接近于给出的频率响应上限，从而使得系统的响应最快。

表 6-11　$F(s)$ 频率响应边界设置表

ω	$M_{\min}(\omega)$	$M_{\max}(\omega)$	$a(\omega)$	$b(\omega)$	$a_F = (a - M_{\min})$	$b_F = (b - M_{\max})$	$\|F(\omega)\|$
0.1	0	0	-0.4	0.4	-0.4	0.4	-0.01
0.5	-0.2	0.1	-2.5	1.5	-2.3	1.4	-0.14
1	1.4	1.1	-5	2	-3.6	0.9	-0.55
2	0.8	1.8	-10	1.5	-9.2	-0.3	-1.9
3	-0.5	1.2	-15.3	0	-14.8	-1.2	-3.7
5	-0.5	1.0	-26	-3	-25.5	-4	-7.6
10	-3.1	1.2	-45	-12	-42	-13	-15.7

由上面给出的前置滤波器可以得出 QFT 控制下不确定系统的闭环响应曲线，如图 6-48 所示，这时 MATLAB 程序段可以如下表示

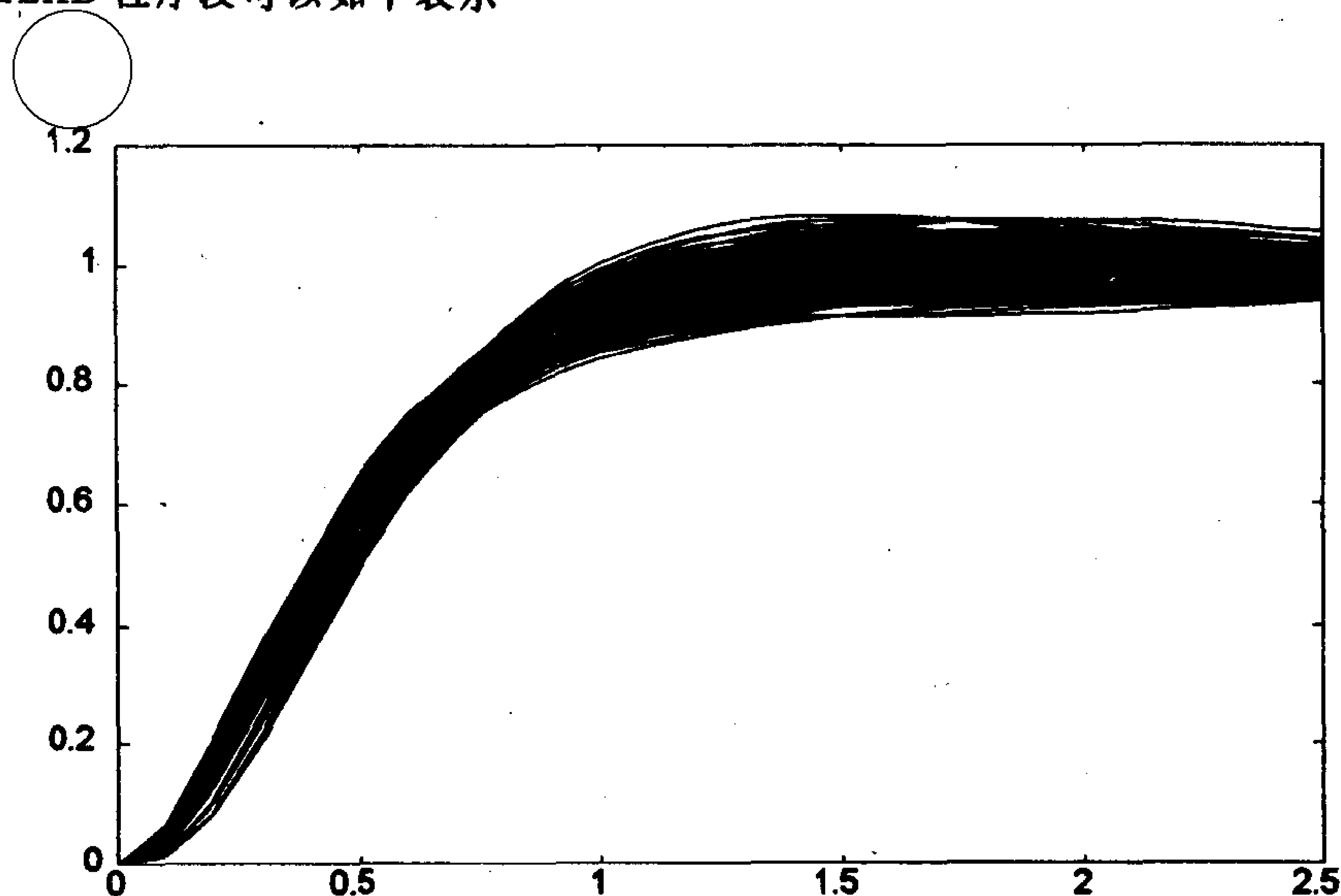


图 6-48　QFT 控制下闭环阶跃响应

```
>> t=0:.1:2.5; yf=[];
```

```
>> for k=1:4
     for A=1:1:4
        for B=-2:1:2
           for C=[1:5, 6.25]
              num=numC*k;
              den=conv([A,B,C],denC)+[zeros(1,3) num];
              den=conv(den,conv([1,3],[1,7])); num=21*num;
              yf=[yf step(num,den,t)];
   end, end, end, end
>> plot(t,yf)
```

6.5.3 QFT 计算机辅助设计工具箱简述

由前面的例子可见， QFT 技术实现起来并不是轻而易举的事。美国学者 Craig Borghesani, Yossi Chait 和 Oded Yaniv 设计了一个基于 MATLAB 语言的通用 QFT 工具箱[8]，为用户根据 QFT 进行控制系统设计提供了有力的工具。由于使用了 MATLAB 的界面设计新函数，所以 QFT 工具箱需要 MATLAB 4.1 及以上版本的支持，在 4.0 版本下有些函数是不可以直接使用的。在这里将简述 QFT 工具箱有关单变量不确定线性系统设计的关键问题：

- **不确定单变量系统的表示方法**: 不确定系统的分子和分母系数可以通过选样的方式来表示，在指定范围内每一个选样的分子和分母都将作为单独的行附加到分子分母模型中，例如例 6.12 中的不确定系统可以由下面 MATLAB 游击队中的矩阵 nump 和 denp 来表示，而标称模型可以由它在 nump 矩阵中的位置序号来表示，如在此例中 nompt=120 表示标称系统位于该矩阵中的第 120 行。

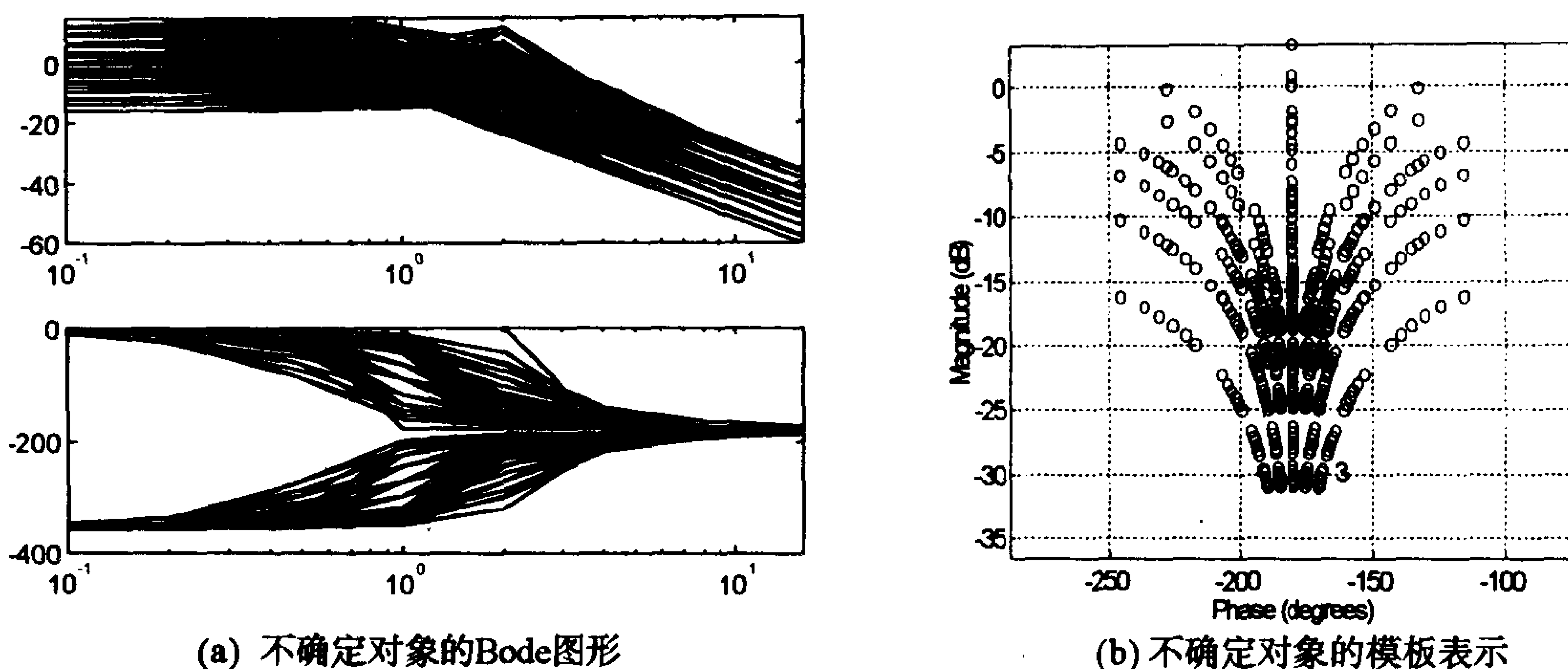
```
>> nump=[]; denp=[];
>>  for k=1:4, for A=1:4, for B=-2:2, for C=[1:1:5 6.25]
       nump=[nump; k];;  denp=[denp; [A,B,C]];
    end, end, end, end
>>  nompt=120;
```

- **不确定系统频率响应构造函数 freqcp()**: 用来求取不确定单变量系统的频率响应数据，该函数的基本调用格式为

```
P=freqcp(w,nump,denp,delay)
```

其中 w 为频率向量，nump 和 denp 分别为不确定系统的分子和分母系数矩阵，delay 为时间延迟矩阵，一般情况下 delay=0 。利用此函数可以绘制出不确定对象模型如图 6-49 (a) 所示的 Bode 曲线。

```
>>  w=[0.1,0.2,0.5,1,2,3,4,8,16];  P=freqcp(nump,denp,w);
>>  subplot(211), semilogx(w,20*log10(abs(P))); axis([0.1,16,-60,15])
>>  subplot(212), semilogx(w,180*qatan4(P)/pi)
```



(a) 不确定对象的Bode图形　　(b) 不确定对象的模板表示

图 6-49　不确定对象模型的 Bode 图及模板表示

注意，在前面的程序段中使用了 qatan4() 函数，它可以用来在 4 个象限内作反正切变换，此函数是 QFT 工具箱提供的，在图 6-49 的相频特性中有两个分支，其中一个起始于 0°，另一个起始于大约 −350°，表明其中前一组对象环节是稳定的而后一组是不稳定的 (即 $B < 0$)。

- **对象模板绘制函数** plottmpl()：若给出不确定系统的频率响应数据，则可以调用 plottmpl() 函数来绘制对象的模板，这些模板是由 o 标号来表示的。该函数的基本调用格式为

```
plottmpl(w, wbnd, P, nom)
```

其中 w 为频率点全集 (行向量)， wbnd 是需要绘制的 w 的子集， P 为不确定对象的频率响应数据矩阵， nom 为标称对象的标号，按照下面格式调用此函数

```
plottmpl(w,3,P,120)
```

就可以绘制出频率为 $\omega = 3$ 时对象的模板来，如图 6-49(b) 所示，得出的 Nichols 曲线和图 6-45(b) 表示的是一致的。

- **单变量系统频率边界函数** sisobnds()：此函数用于求取边界信息，其调用格式为

```
bnds=sisobnds(ptype,w,wbnd,W2,P,R,nom))
```

其中 ptype 为求解类型 (其中 ptype=1 表示求解稳定裕度下的频率响应边界)， w，wbnd，P，nom 的定义和前面一致， W2 为稳定裕度要求， R 在 ptype=1 时可以为空矩阵。获得了稳定边界之后，就可以调用 plotbnds(bnds) 将稳定边界绘制出来。

- **QFT 设计的图形界面程序** lpshape()：此函数为 QFT 中的图形用户界面表示，其基本调用格式为

```
lpshape(w,bnd,nL0,dL0,delay,nc0,dc0)
```

其中 w 为频率向量，bnd 为计算出来的边界，nL0，dL0 分别为标称对象的分子和分母多项式，delay 为标称对象的延迟系数，nc0，dc0 分别为初始控制器的分子和分母多项式。该函数将给出如图 6-50 所示的图形界面，允许用户设计出 QFT 控制器，并进行相应的处理。

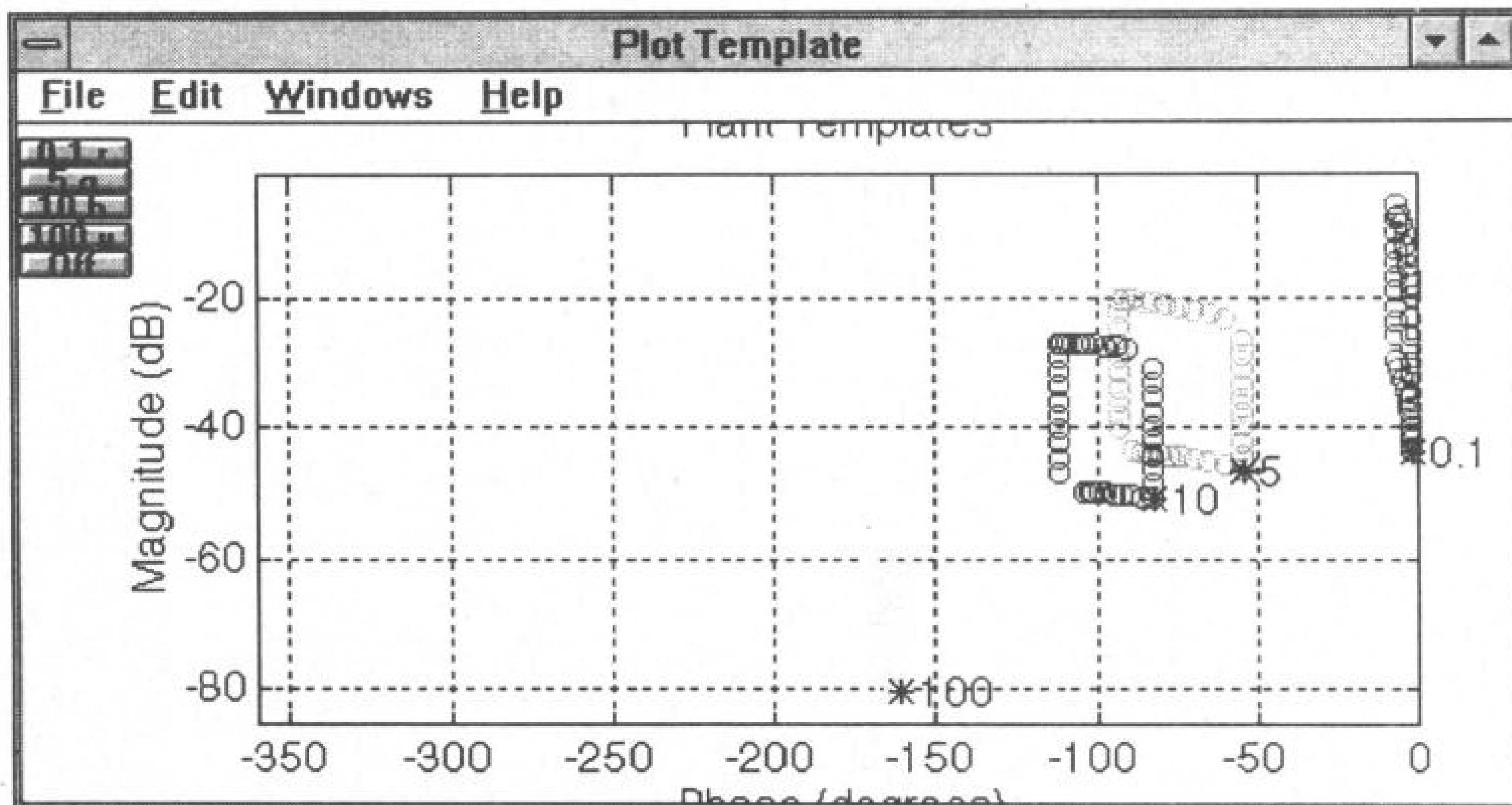


图6-50　QFT 控制器设计图形界面

习　　题

1) 求出状态方程模型 (A,B,C,D) 的传输零点，其中

$$A=\begin{bmatrix}1&1&0\\0&1&0\\0&1&1\end{bmatrix},\ B=\begin{bmatrix}0&1\\1&0\\0&1\end{bmatrix},\ C=[1,1,1],D=[0,0]$$

2) 若传递函数 $T(s)=P^{-1}(s)Q(s)$, 其中

$$P(s)=\begin{bmatrix}2s^2-2&2s^2+1\\-s&-s-3\end{bmatrix},\ Q(s)=\begin{bmatrix}s^3-s^2+s-1\\s+1\end{bmatrix}$$

估算该系统的最小阶次，并求出 $T(s)$ 的 Smith-McMillan 标准型，验证其最小阶次。

3) 考虑下面给出的 2 输入 2 输出系统

$$G(s)=\begin{bmatrix}\dfrac{0.806s+0.264}{s^2+1.15s+0.202}&\dfrac{-(15s+1.42)}{s^3+12.8s^2+13.6s+2.36}\\\dfrac{1.95s^2+2.12s+4.90}{s^3+9.15s^2+9.39s+1.62}&\dfrac{7.14s^2+25.8s+9.35}{s^4+20.8s^3+116.4s^2+111.6s+188}\end{bmatrix}$$

绘制出带有 Gershgorin 带的逆 Nyquist 曲线，并在该曲线上标出各个频率下的特征值，验证这些特征值满足 Gershgorin 定理。

4) 对上面的系统依照逆 Nyquist 方法设计出控制器来。

5) 给出下面的多变量系统传递函数矩阵

$$G(s)=\frac{1}{d(s)}\begin{bmatrix} 29.20s+263.3 & -(3.146s^3+32.67s^2+89.83s+31.81) \\ 5.679s^3+42.67s^2-68.84s-106.8 & 9.430s+15.15 \end{bmatrix}$$

其中 $d(s)=s^4+11.67s^3+15.75s^2-88.31s+5.514$, 试在 $\omega=0,\omega=1,\omega=10$ 等各个频率处分别对该系统作伪对角化，并绘制出各个频率处的逆 Nyquist 图和 Gershgorin 带，比较对角化的效果。

6) 对得出的对角占优矩阵设计反馈控制参数 f_i, 并绘制出 Ostrowski 带来观察设计效果。

7) 用 MATLAB 编写出多点频率加权方式下的伪对角化函数。

8) 给定多变量传递函数矩阵 $G(s)$ 为

$$G(s)=\begin{bmatrix} \dfrac{0.8(s-1)}{(s+1)(s+2)} & \dfrac{0.8s}{(s+1)(s+2)} \\ \dfrac{-4.8}{(s+1)(s+2)} & \dfrac{0.8(s-2)}{(s+1)(s+2)} \end{bmatrix}$$

试绘制出系统的特征轨迹图形，并判定系统的稳定性，并设计出近似可交换的控制器。

9) 考虑 5) 中的 2×2 系统模型，试用特征轨迹法设计出控制器，并和习题 5) 比较其效果。

10) 已知系统的模型为

a) $G_1(s)=\dfrac{2(0.5s+1)e^{-0.1s}}{(s+1)(4s+1)}$, b) $G_2(s)=\dfrac{1}{(s+1)^5}$, c) $G_3(s)=\dfrac{1.2e^{-10s}}{(5s+1)(2.5s+1)}$

试利用给出的各种方法对之进行 PID 控制，并比较结果。

11) 使用 MATLAB 语言设计出最优 PID 自整定算法及幅值相位方法得出的 PID 控制器参数的函数。

12) 考虑对前面的 2 阶不确定系统编写 QFT 的 MATLAB 通用程序，若给定的传递函数为 $G_1(s)=1/(s+a)$, $G_2(s)=k(s+b)/(s+a)$, 其中不确定参数的取值范围为 $a\in[-2,2]$, $b\in[-1,2], k\in[1,3]$, 试利用 QFT 方法分别设计出控制器，并比较阶跃响应与频率响应。

参 考 文 献

[1] Åström K J, Hägglund T. Automatic tuning of simple regulators with specification on phase and amplitude margins. Automatica, 1984, 20: 645-651

[2] Åström K J, Hägglund T. Automatic tuning of PID controllers. Research Triangle Park N.C.: Instrument Society of America, 1988

[3] Åström K J, Hang C C, Persson P, Ho W K. Towards intellegient PID control. Automatica, 1992, 28(1): 1-9

[4] Åström K J, Wittenmark B. Adaptive control. Reading: Addison-Wesley, 1989

[5] Atherton D P. Nonlinear control engineering — describing function analysis and design. London: Van Nostrand Reinhold, 1975

[6] Balasubramanian R. Continuous time controller design. IEE Control Engineering Series Vol. 39. London: Peter Peregrinus Ltd, 1989

[7] Bennett S. Development of the PID controllers. IEEE Control Systems Magazine, 1993, 13(2): 58-65

[8] Borghesani C, Chait Y, Yaniv O. QFT toolbox user's manual. MathWorks, 1993

[9] Boyel J M, Ford M P, Maciejowski J M. A multivariable toolbox for use with MATLAB. IEEE Control Systems Magazine, 1989, 9: 59-65

[10] Edmunds J M. Control system design and analysis using closed-loop Nyquist and Bode arrays. Int. J. Control, 1979, 30: 773-802

[11] Ford M P, Daly K C. Dominance improvement by pseudodecoupling. Proceedings IEE. Pt D, 1979, 126: 1316-1320

[12] 韩京清. 一种新型控制器— NLPID. 控制与决策, 1994, 9(6): 401-407

[13] Hang C C, Åström K J, Ho W K. Refinement of the Ziegler-Nichols tuning formula. Proceedings IEE. Pt D, 1991, 138: 111-118

[14] Horowitz I. Quantitative feedback theory (QFT). Proceedings IEE. Pt D, 1982, 129: 215-226

[15] Horowitz I. Survey of quantitative feedback theory (QFT). Int. J. Control, 1991, 53(2): 255-291

[16] Horowitz I. Lecture notes for the workshop on quantitative feedback theory (QFT). 30th IEEE CDC. Brighton. UK, 1991

[17] Horowitz I. Quantitative feedback design theory. 1992

[18] Hung Y S, MacFarlane A G J. Multivariable feedback: a quasi-classical approach. Lecture Notes in Control and Information Sciences. Vol. 40. New York: Springer-Verlag, 1982

[19] Hwakins D J. Pseudodiagonalisation and the inverse Nyquist array method. Proceedings IEE. Pt D, 1972, 119: 337-342

[20] Kouvaritakis B. Theory and practice of the characteristic-locus design method. Proceedings IEE. Pt D, 1979, 126: 542-548

[21] MacFarlane A G J. The return-difference and return-ratio matrices and their use in the analysis and design of multivariable feedback control systems. Proceedings IEE. Pt D, 1970, 117: 2037-2049

[22] MacFarlane A G J, Kouvaritakis B. A design technique for linear multivariable feedback systems. Int. J. Control, 1977, 25: 837-874

[23] MacFarlane A G J, Scott-Jones D F A. Vector gain. Int. J. Control, 1979, 29: 65-91

[24] Maciejowski J M. Multivariable feedback design. Wokingham: Addison-Wesley, 1989

[25] Mayne D Q. Sequential design of linear multivariable systems. Proceedings IEE, Pt D, 1979, 126: 568-572

[26] Munro N. Multivariable control 1: the inverse Nyquist array design method. Lecture notes on SERC vacation school on Control System Design. UMIST, Manchester, 1989

[27] Patel R V, Munro N. Multivariable system theory and design. Oxford: Pergamon Press, 1982

[28] Phalen R M. Automatic control systems. Cornell: Cornell University Press, 1977

[29] Rosenbrock H H. Computer aided control systems design. London: Academic Press, 1973

[30] Tsypkin Ya Z. Relay control systems. Cambridge: Cambridge University Press, 1984

[31] Xue D, Atherton D P. A menu-driven model reduction program and its applications. In: Barker H A, (ed). Computer-aided design in control systems. Oxford: Pergamon Press, 1992. 232-238

[32] Xue D, Atherton D P. A reduction-based PDF controller design algorithm. Proceedings IEEE Conference on Decision and Control. Texas, USA, 1993, 2917-2922,

[33] Xue D, Atherton D P. A suboptimal reduction algorithm for linear systems with a time delay. Int. J. Control, 1994, 60(2): 181-196

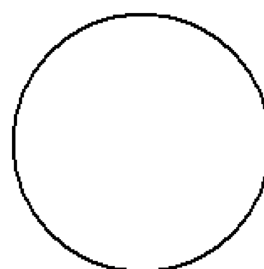
[34] Zakian V. New formulation for the method of inequalities. Proceedings IEE, Pt D, 1979, 126: 579-584

[35] Zakian V, Al-Naib U. Design of dynamical and control systems by the method of inequalities. Proceedings IEE, Pt D, 1973, 120: 1421-1427

[36] Zhuang M. Computer aided PID controller design. D.Phil. thesis. the University of Sussex, Brighton, U.K, 1992

[37] Zhuang M, Atherton D P. Automatic tuning of optimum PID controllers. Proceedings IEE, Pt D, 1993, 140: 216-224

[38] Ziegler J G, Nichols N B. Optimum settings for automatic controllers. Transaction of ASME, 1942, 64: 759-768





第 7 章 控制系统计算机辅助设计 (时域方法)

7.1 引　言

与前面介绍的频域设计方法同时发展的时域设计技术，基于状态空间模型的设计方法在近 30 年来也有了很大的进展。50 至 60 年代，控制系统的研究者们提出了各种控制系统设计方法，其中比较有代表性的包括极点配置设计方法、多变量系统的解耦控制、模型跟踪系统的设计方法以及以线性二次型性能 (LQ) 指标为基础的最优控制问题，后来由于考虑状态量测与扰动等问题而发展成求解线性二次型 Gauss (LQG) 问题，在这方面有较丰富的文献资料，如文献 [3] 等。事实上单纯的 LQG 存在很多问题，这也就引出了回路传输恢复 (LTR) 技术的研究 [23]。

在 80 年代后期发展起来的 H_∞ 最优控制策略更趋于理论化，利用 MATLAB 的鲁棒控制工具箱和 μ 综合与控制工具箱可以相对容易地解决这些鲁棒控制系统的设计问题，但使用起来需要一定的理论基础。

值得指出的是，时域设计方法和频域设计方法并不是相互独立而发展的，它们之间有着千丝万缕的联系，因为毕竟它们是描述同一控制系统的，而它们之间又有等效关系。例如在进行理论证明时往往需要从状态空间的角度去完成，而控制器等问题又往往需要在频域下来实现。

7.2 基于状态反馈的经典设计方法

7.2.1 极点配置与模态控制

给定控制系统的状态方程模型，则经常希望引入某种控制器，使得该系统的闭环极点移动到某个指定位置，因为在很多情况下系统的极点位置会决定系统的动态性能。在这里将只考虑单变量系统的极点配置问题。假设系统的状态方程模型为

$$\dot{x} = Ax + bu,\ y = cx \tag{7.2.1}$$

其中 A, b, c 具有兼容的维数，这时若引入状态反馈，使得进入该系统的信号为 $u = r - k^T x$，这里 r 为系统的外部参考输入， k 为列向量，这时闭环系统的状态方程模型可以写成

$$\dot{x} = (A - bk^T)x + br,\ y = cx \tag{7.2.2}$$

并可以证明，若给定的系统是可控的，则可以通过状态反馈将该系统的闭环极点进行任意地配置。下面将介绍几种极点配置的常用算法

- Bass-Gura **配置算法** [5, 6]: 假设期望的闭环系统极点为 $\mu_i, i=1,\cdots,n$, 则原系统开环特征方程 $a(s)$ 和闭环系统的特征方程 $\alpha(s)$ 分别可以写成

$$a(s)=\det(sI-A)=s^n+\sum_{i=0}^{n-1}a_is^i,\ \alpha(s)=\prod_{i=1}^{n}(s-\mu_i)=s^n+\sum_{i=0}^{n-1}\alpha_is^i \tag{7.2.3}$$

若原系统为可控的，则可以由下式求出系统的状态反馈向量 k

$$k^T=(\alpha-a)^TL^{-1}\mathcal{C}^{-1} \tag{7.2.4}$$

式中， $(a-\alpha)^T=[(a_0-\alpha_0),\cdots,(a_{n-1}-\alpha_{n-1})]$, $\mathcal{C}=[b,Ab,\cdots,A^{n-1}b]$ 且

$$L=\begin{bmatrix} a_1 & a_2 & \cdots & a_{n-1} & 1\\ a_2 & a_3 & \cdots & 1 & \\ \vdots & \vdots & \ddots & & \\ a_{n-1} & 1 & & & \\ 1 & & & & \end{bmatrix} \tag{7.2.5}$$

可见若原系统可控，则 $\mathcal{C}^{-1}$ 存在，另外显见 L 为非奇异 Hankel 矩阵，故可以任意地配置原系统。

- Ackermann **配置算法**: 问题描述和前面一致，但解决方法略有不同，由 Ackermann 极点配置算法可以如下构造出状态反馈向量 k

$$k^T=-[0,0,\cdots,0,1]\mathcal{C}^{-1}\alpha(A) \tag{7.2.6}$$

- **部分极点配置**: 在一些特定的应用中有时没有必要去对所有的极点进行重新配置，而只需对其中若干个极点进行配置使得其它极点保持原来的值，例如若系统开环模型是不稳定的，则可以将那些不稳定的极点配置成稳定的值，而不去改变那些原本稳定的极点。作这样配置的前提条件是原系统没有重极点，这就能保证由系统特征向量构成的矩阵是非奇异的。假设 x_i 为对应于 λ_i 的特征向量，即 $Ax_i=\lambda_ix_i$, 这样可以对各个特征值构造出特征向量矩阵 $X=[x_1,\cdots,x_n]$, 由前面的假设可知 X 矩阵为非奇异的，故可以得出其逆阵 $T=X^{-1}$, 且令 T 的第 i 个行向量为 t_i, 且想把 λ_i 配置到 μ_i 的位置，则可以定义变量 $\gamma_i=(\mu_i-\lambda_i)/b_i$, 其中 b_i 为向量 Tb 的第 i 个分量，这时配置全部的极点，则可以得出状态反馈向量

$$k^T=\sum_{i=1}^{n}\gamma_it_i \tag{7.2.7}$$

特别地，若不想对哪个极点进行重新配置，则可以将对应的项从上面的求和式子中剔除就可以得出相应的状态反馈向量，它能按照指定的方式进行极点配置。

例 7.1 考虑下面的开环状态方程模型

$$\dot{x}=\begin{bmatrix}-2&-1&1\\1&0&1\\-1&0&1\end{bmatrix}x+\begin{bmatrix}1\\1\\1\end{bmatrix}u$$

可以由 MATLAB 命令输入原系统，并得出系统的特征值来

```
>> A=[-2,-1,1; 1,0,1; -1,0,1]; b=[1;1;1];
>> eig(A)
ans =  -1.0000 + 1.0000i
       -1.0000 - 1.0000i
        1.0000
```

可见原系统有不稳定极点。下面由 MATLAB 命令分别采用 Bass-Gura 算法和 Ackermann 算法将闭环系统的极点配置到 $-1, -2, -3$ 。

```
>> CharA=poly(A); CharN=conv([1,1],conv([1,2],[1,3])); ii=length(CharA):-1:2;
>> diffA=CharA(ii)-CharN(ii); CC=b; bb=b;
>> for i=2:length(A), bb=A*bb; CC=[CC bb]; end
>> k1=diffA*inv(hankel([CharA(length(CharA)-1:-1:2)';1]))*inv(CC)
k1 =  1.0000    -2.0000    -4.0000
>> eig(A+b*k1)'
ans =  -2.0000   -3.0000   -1.0000
>> k2=-[zeros(1,length(A)-1), 1]*inv(CC)*polyvalm(CharN,A)
k2 =     1      -2      -4
```

可见采用这样两种方法会得出相同的结果。并可以看出采用 Ackermann 算法会以更高的精度得出状态反馈向量。其实原系统只有一个不稳定极点，故可以采用后面给出的算法对该极点进行配置，例如配置到 -5, 则可以给出下面的命令

```
>> [T,dd]=eig(A); i=find(diag(dd)>0); iT=inv(T); mu=-5; v=diag(dd);
>> bb=iT*b; gamma=(mu-v(i))/bb(i);
>> k3=real(gamma*iT(i,:))
k3 =  1.5000   -1.5000   -6.0000
>> eig(A+b*k3)'
ans = -5.0000 + 0.0000i  -1.0000 - 1.0000i  -1.0000 + 1.0000i
```

MATLAB 的控制系统工具箱提供了极点配置的函数 place() 和 acker(), 其调用格式是相同的

k=place(A,b,p)　或　k=acker(A,b,p)

其中 p 为指定极点的位置。调用这样两个函数也可以得出和前面同样的结果。

```
>> p=[-1; -2; -3]; k11=place(A,b,p')
place: ndigits= 17
k11 =    -1.0000     2.0000     4.0000
>> k12=acker(A,b,p')
k12 =    -1       2       4
```

多变量系统的极点配置问题在这里就不加讨论了，可以参照文献[16], 事实上 MATLAB 控制系统工具箱提供的 place() 函数可以直接处理多变量系统的极点配置问题，下面将通过例子来演示该函数调用及配置结果。

例 7.2　假设双输入线性系统的状态方程为 [5]

$$\dot{x}=\begin{bmatrix}0&0&4&1\\10&13&2&8\\-3&-3&0&-2\\-10&-14&-5&-9\end{bmatrix}x+\begin{bmatrix}-2&0\\4&-3\\-1&1\\-3&3\end{bmatrix}u$$

若想引入状态反馈矩阵 K，使得闭环系统的极点位置为 $-2,-3,(-1\pm\sqrt{-3})/2]$，则可以使用下面的 MATLAB 命令来完成

```
>>  A=[0 0 4 1;10 13 2 8; -3 -3 0 -2; -10 -14 -5 -9]; B=[-2 0; 4 -3; -1 1; -3 3];
>>  P=[-2;-3;(-1+sqrt(-3))/2; (-1-sqrt(-3))/2];
>>  K=acker(A,B,P)
??? Error using ==> acker
System must be single input
>> K=place(A,B,P)
place: ndigits= 18
K = -116.9276 -230.5121 -226.7158 -160.3533
    -113.6643 -222.1870 -215.2101 -153.6445
>>  P1=eig(A-B*K)'
P1 =  -3.0000              -0.5000 - 0.8660i  -0.5000 + 0.8660i  -2.0000
>>  norm(sort(P1)-sort(P))
ans =  1.7399e-011
```

可见这里的 Ackermann 算法不能处理多变量系统，而 place() 函数则可以成功地对多变量系统作任意的极点配置。其实极点配置的矩阵 K 并不是唯一的，例如文献 [5] 中给出了其它几个状态反馈矩阵

$$K_1=\begin{bmatrix}12 & 29 & 33 & 17\\ 6 & 15 & 17 & 10\end{bmatrix},\quad K_2=\begin{bmatrix}-3 & -6 & -9 & -4\\ 82 & 183 & 202 & 118\end{bmatrix} \tag{7.2.8}$$

它们也可以达到同样的极点配置目的，但设计出来的闭环系统性能却不相同，以第 1 状态变量 $x_1(t)$ 的曲线为例可以看出各种配置下阶跃响应是大不相同的，如图 7-1 所示。所以说，单靠极点配置技术并不一定能够设计出满意的控制器来。

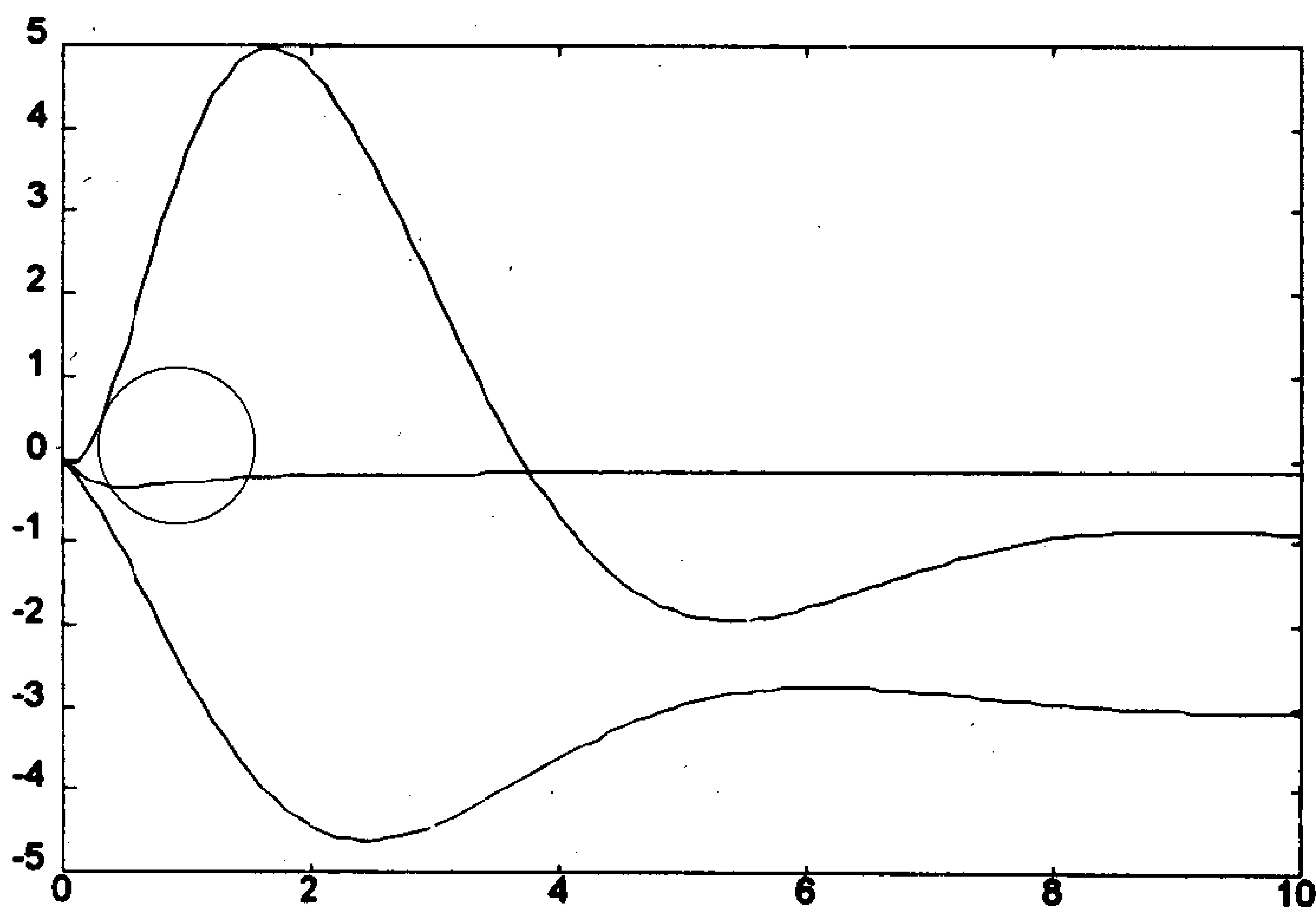


图7-1　不同状态反馈矩阵下的阶跃响应比较

```
>>  K1=[12 29 33 17; 6 15 17 10]; K2=[-3 -6 -9 -4; 82 183 202 118]; t=0:0.1:10;
>>  C=[1 0 0 0]; y=step(A-B*K,B,C,[0 0],1,t);
>>  y1=step(A-B*K1,B,C,[0 0],1,t); y2=step(A-B*K2,B,C,[0 0],1,t);
>>  plot(t,[y y1 y2])
```




7.2.2 解耦控制

第 6 章中介绍的传递函数矩阵对角化的方法从某种意义上来讲就是解耦控制 (decoupling control)，在多变量系统中，不同的输入与输出之间存在着耦合，即第 1 输入不但会对第 1 输出有影响，而且还会影响到其它的输出，这就给控制系统的设计造成了很大的麻烦。这样很自然地在多变量控制系统的设计中就出现了解耦控制方法 [12, 20]，假设已知控制系统的状态方程模型为

$$\dot{x} = Ax + Bu, \quad y = Cx + Du \tag{7.2.9}$$

其中状态变量个数为 n, 而输入输出的个数分别为 p 和 m, 这里 A, B, C, D 为兼容维数的矩阵。引入状态反馈 $u = Gr - Kx$, 其中 r 为 $m \times 1$ 参考输入向量，在解耦控制中实际还应该要求 $p = m$, 亦即系统的输入个数等于输出个数，这时闭环系统的传递函数矩阵可以写成

$$Y(s) = \Big[(C - DK)(sI - A + BK)^{-1}B + D\Big]GR(s) = \Psi(s)R(s) \tag{7.2.10}$$

这里 $\Psi(s)$ 是一个 $m \times m$ 的传递函数矩阵。

若闭环系统的 $\Psi(s)$ 矩阵为对角的非奇异矩阵，则称该系统是动态解耦的系统，若 $\Psi(0)$ 为对角非奇异矩阵，且系统为稳定的，则称该系统是静态解耦的。

在给定的控制结构下，若系统的 D 矩阵为 0，则闭环传递函数矩阵 $\Psi(s)$ 可以简化成

$$\Psi(s) = (sI - A + BK)^{-1}BG = C(sI - A_1)^{-1}BG, \quad \text{其中} \quad A_1 = A - BK \tag{7.2.11}$$

由上式可见，若 G 矩阵为奇异矩阵，则 $\Psi(s)$ 矩阵必须为奇异的，所以为使得系统可以解耦首先应该要求 G 为非奇异矩阵。对给定的系统状态方程可以写出

$$\Psi(s) = \frac{1}{\det(sI - A_1)}\Big[CBs^{n-1} + (CA_1B + a_{n-1}CB)s^{n-2} + \cdots\Big]G \tag{7.2.12}$$

其中分母多项式可以写成 $\det(sI - A_1) = s^n + \sum_{i=0}^{n-1} a_i s^i$ 。

首先在这里将给出可解耦的条件：可以证明，若按下面方法生成的矩阵 B^* 为非奇异的，则由前面给出的控制格式得出的系统可以解耦原系统。

$$B^* = \begin{bmatrix} c_1^T A^{d_1} B \\ \vdots \\ c_m^T A^{d_m} B \end{bmatrix} \tag{7.2.13}$$

这时若选择了 d_i 参数，则可以直接获得解耦 K 矩阵

$$K = B^{*-1} \begin{bmatrix} c_1^T A^{d_1+1} \\ \vdots \\ c_m^T A^{d_m+1} \end{bmatrix} \tag{7.2.14}$$

例 7.3 考虑给出的双输入双输出系统

$$\dot{x}=\begin{bmatrix}-1&0&0\\0&-2&0\\0&0&-3\end{bmatrix}x+\begin{bmatrix}1&0\\2&3\\-3&-3\end{bmatrix}u,\quad y=\begin{bmatrix}1&0&0\\1&1&1\end{bmatrix}x$$

可以由下面的命令得出系统的传递函数矩阵

```
>>  A=[-1,0,0; 0,-2,0; 0,0,-3]; B=[1,0; 2,3; -3,-3]; C=[1,0,0; 1,1,1]; D=zeros(2,2);
>>  [nn1,dd1]=ss2tf(A,B,C,D,1)
nn1 =      0     1.0000     5.0000     6.0000
           0     0.0000     4.0000     6.0000
dd1 =      1       6        11         6
>>  [nn2,dd2]=ss2tf(A,B,C,D,2)
nn2 =      0       0        0          0
           0       0        3          3
dd2 =      1       6        11         6
```

亦即系统的传递函数矩阵可以写成

$$G(s)=\frac{1}{s^3+6s^2+11s+6}\begin{bmatrix}s^2+5s+6&0\\4s+6&3s+3\end{bmatrix}$$

使用 MATLAB 命令可以构造出 B^* 矩阵，并得出 d_i 参数，最后构造出解耦控制器 G 和 K 来

```
>>  [m,n]=size(C); BB=C(1,:)*B; d(1)=0;
>>  for i=2:m
       for j=0:n-1
          BB=[BB; C(i,:)*A^j*B];
          if rank(BB)==i, d(i)=j; break; else, BB=BB(1:i-1,:); end
    end, end
>>  G=inv(BB)
G =    1.0000        0
      -1.3333     0.3333
>>  CC=C(1,:)*A^(d(1)+1);for i=2:m, CC=[CC; C(i,:)*A^(d(i)+1)]; end; K=G*CC
K =   -1.0000        0          0
       1.6667     1.3333     3.0000
>>  B1=B*G; A1=A-B*K; [n1,d1]=ss2tf(A1,B1,C,D,1), [n2,d2]=ss2tf(A1,B1,C,D,2)
n1 =         0     1.0000        0          0
             0     0.0000     0.0000     0.0000
d1 =         1       0          0          0
n2 =         0       0          0          0
             0       0          1          0
d1 =         1       0          0          0
```

亦即系统的变换传递函数矩阵为

$$\Psi(s)=\frac{1}{s^3}\begin{bmatrix}s^2&0\\0&s\end{bmatrix}=\frac{1}{s^2}\begin{bmatrix}s&0\\0&1\end{bmatrix}$$

解耦控制的目的是将原模型变换成解耦的模型，而并不必去考虑变换之后的响应品质，因为响应品质这类问题可以在解耦之后按照单回路进行设计补偿，最后建立其如图

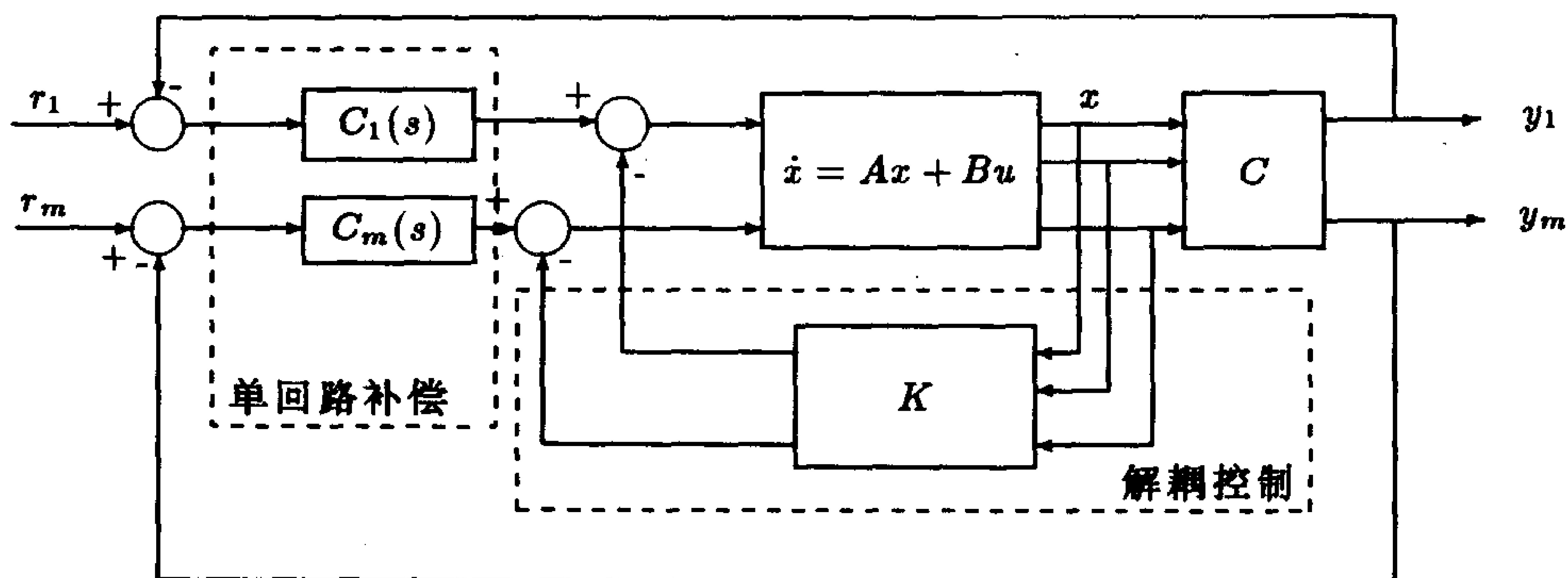


图 7-2 解耦控制框图结构

7-2 所示的控制框图，单回路的设计当然可以采用各种方法，例如可以采用超前滞后补偿、PI 设计以及 PID 设计等，并能保证这样设计出来的控制器不会去影响其它回路。

例 7.4 下面可以考虑一个 3×3 模型的解耦，已知线性 LTI 系统的状态方程模型参数 (A,B,C) 为

$$A=\begin{bmatrix}0 & 0 & 1.1320 & 0 & -1\\ 0 & -0.0538 & -0.1712 & 0 & 0.0705\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0.0485 & 0 & -0.8556 & -1.013\\ 0 & -0.2909 & 0 & 1.0532 & -0.6859\end{bmatrix},\ B=\begin{bmatrix}0 & 0 & 0\\ -0.120 & 1 & 0\\ 0 & 0 & 0\\ 4.419 & 0 & -1.665\\ 1.575 & 0 & -0.0732\end{bmatrix},\ C=\begin{bmatrix}I_3 & 0_{3\times 2}\end{bmatrix}$$

通过类似前面的分析，也可以假定 $d_1=0$，但这时会发现 $c_1A^{d_1}B\equiv 0$，故这里分析应该从 $d_1=1$ 开始，重复上面的过程就可以得出系统的解耦结果

```
>> A=[0,0,1.1320,0,-1; 0,-0.0538,-0.1712,0,0.0705; 0,0,0,1,0;
      0,0.0485,0,-0.8556,-1.013; 0,-0.2909,0,1.0532,-0.6859];
>> B=[0,0,0; -0.120,1,0; 0,0,0; 4.419,0,-1.665; 1.575,0,-0.0732];
>> C=[eye(3,3)zeros(3,2)]; D=zeros(2,2);
>> [m,n]=size(C); BB=C(1,:)*A*B; d(1)=1;
>> for i=2:m
      for j=0:n-1
         BB=[BB; C(i,:)*A^j*B];
         if rank(BB)==i, d(i)=j; break; else, BB=BB(1:i-1,:); end
   end, end
>> d
d =   1     0     1
>> G=inv(BB); CC=C(1,:)*A^(d(1)+1);for i=2:m, CC=[CC; C(i,:)*A^(d(i)+1)]; end;
>> K=G*CC
K =   0   -0.2122         0   -0.0298   -0.4645
      0   -0.0793   -0.1712   -0.0036    0.0148
      0   -0.5924         0    0.4347   -0.6244
```

同样可以由下面语句得出系统的传递函数矩阵

```
>> B1=B*G; A1=A-B*K;  [n1,d1]=ss2tf(A1,B1,C,D,1)
n1 =        0        0    1.0000    0.0000    0.0000    0.0000
```

```
           0    0.0000    0.0000         0         0         0
           0    0.0000    0.0000    0.0000         0         0
d1 = 1.0000    0.0000    0.0000         0         0         0
>>  [n2,d2]=ss2tf(A1,B1,C,D,2), [n3,d3]=ss2tf(A1,B1,C,D,3)
n2 =       0    0.0000    0.0000    0.0000    0.0000         0
           0    1.0000    0.0000         0         0         0
           0    0.0000    0.0000    0.0000         0         0
d2 = 1.0000    0.0000    0.0000         0         0         0
n3 =       0    0.0000    0.0000    0.0000    0.0000         0
           0    0.0000    0.0000         0         0         0
           0         0    1.0000    0.0000         0         0
d3 = 1.0000    0.0000    0.0000         0         0         0
```

亦即系统的传递函数矩阵为

$$G(s)=\frac{1}{s^5}\begin{bmatrix} s^3 & & \\ & s^4 & \\ & & s^3 \end{bmatrix}=\frac{1}{s^2}\begin{bmatrix} s^2 & & \\ & s & \\ & & s^2 \end{bmatrix}$$

可见引入状态反馈矩阵 K 后原系统可以完全解耦。

文献 [5, 24] 还讨论了静态解耦的问题，在这里就不再叙述了。

7.2.3 模型跟踪控制

模型跟踪控制是使得系统的输出逼近给定模式的一种控制方案，其方法是强迫跟踪误差 (即系统的输出与参考模型的输出之差) 趋近于 0，以达到控制的目的 [11]。假设原始系统的模型为

$$\dot{x}_p=A_px_p+B_pu_p,\quad y_p=C_px_p \tag{7.2.15}$$

其中 x_p, u_p 和 y_p 分别为 n,r,m 维列向量，且参考模型为

$$\dot{x}_m=A_mx_m+B_mu_m,\quad y_m=C_mx_m \tag{7.2.16}$$

其中 x_m, u_m 和 y_m 分别为 n_1,r_1,m 维列向量。下面将分两种情况来考虑模型跟踪控制的设计方法：

- **参考模型阶次等于系统阶次**：这时可以定义一个跟踪误差向量 $e=x_m-x_p$, 使得

$$\dot{e}=A_me+(A_m-A_p)x_p+B_mu_m-B_pu_p \tag{7.2.17}$$

这样可以选择下面的控制律

$$u_p=B_p^+(A_m-A_p)x_p+B_p^+B_mu_m=K_px_p+K_uu_m \tag{7.2.18}$$

式中 B_p^+ 为 B_p 矩阵的伪逆，所以若 A_m 稳定且能满足下面的条件

$$(I-B_pB_p^+)(A_m-A_p)=0,\quad (I-B_pB_p^+)B_m=0 \tag{7.2.19}$$

则 $\dot{e}=A_me$, 即可以使得跟踪误差渐近地趋近于 0。

- **参考模型阶次不等于系统阶次**：选择参考模型的阶次等于系统的输出个数，即 $n_1 = m$，这时可以定义一个误差向量 $e = x_m - y_p$，使得

$$\dot{e} = A_m e + (A_m C_p - C_p A_p)x_p + B_m u_m - C_p B_p u_p \tag{7.2.20}$$

这样可以选择下面的控制律

$$u_p = (C_p B_p)^+ (A_m C_p - C_p A_p)x_p + (C_p B_p)^+ B_m u_m = K_p x_p + K_u u_m \tag{7.2.21}$$

所以若 A_m 稳定且能满足下面的条件

$$[I - (C_p B_p)(C_p B_p^+)](A_m C_p - C_p A_p) = 0, \quad [I - (C_p B_p)(C_p B_p^+)]B_m = 0 \tag{7.2.22}$$

则 $\dot{e} = A_m e$，即可以使得跟踪误差渐近地趋近于 0 。

这种控制策略又称为隐含模型跟踪控制 (implicit model-following control)，其控制结构图如图 7-3 所示，因为在此结构图中参考模型并没有显式地加以实现，而只是在控制律的获取时才能体现出来。

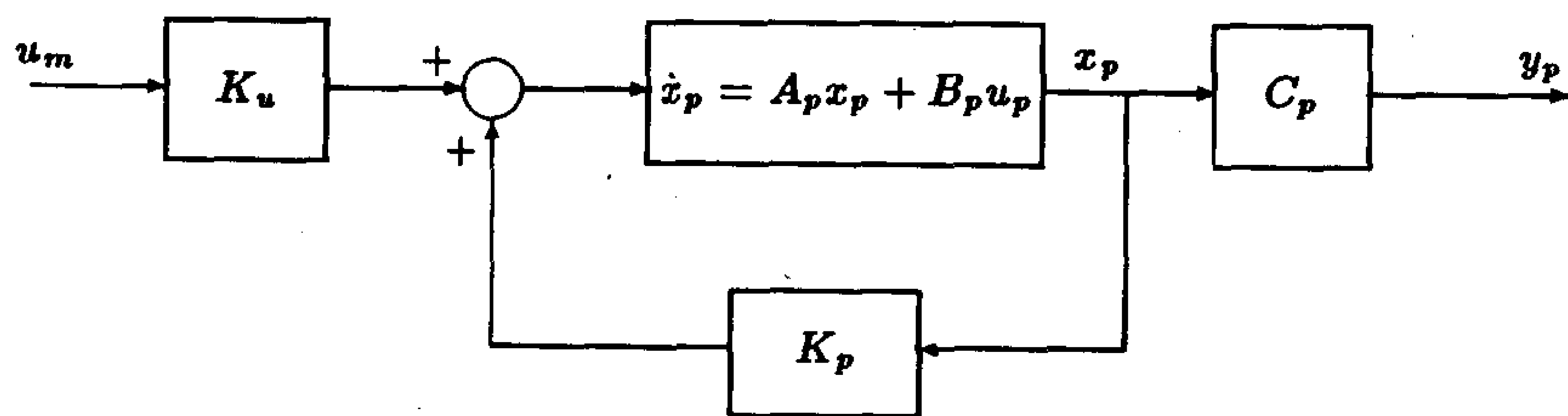


图 7-3　隐含模型跟踪控制结构图

再考虑下面一种控制方案，其结构图如图 7-4 所示，因为在此结构图中显式地实现

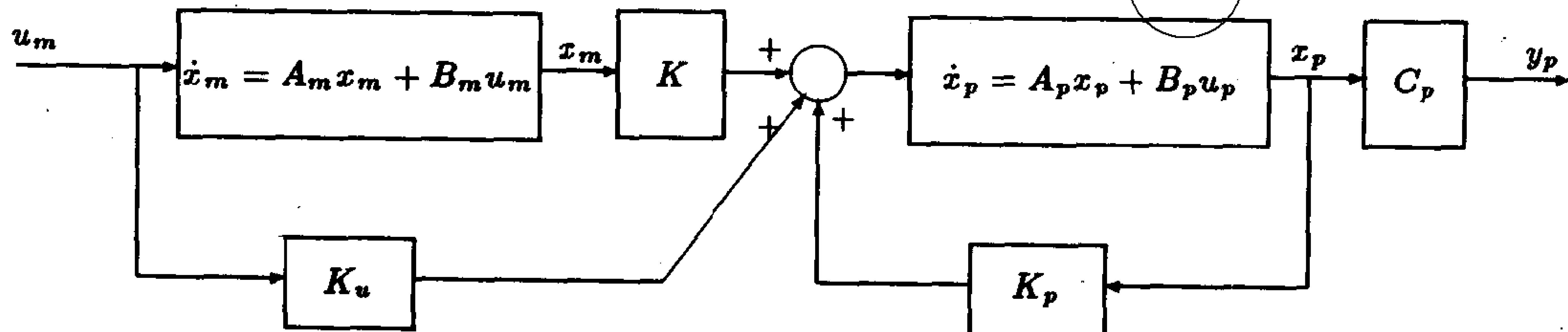


图 7-4　实际模型跟踪控制结构图

了参考模型，所以又称为实际模型跟踪控制 (real model-following control)，这里仍分下列两种情况加以考虑：

- **参考模型阶次等于系统阶次**：仍定义 $e = x_m - x_p$，并选择

$$u_p = u_1 + u_2, \quad 其中 \quad u_1 = Ke, \; u_2 = B_p^+(A_m - A_p)x_p + B_p^+ B_m u_m \tag{7.2.23}$$

这时

$$u_p = Kx_m + K_p x_p + K_u u_m, \text{ 其中 } K_p = B_p^+(A_m - A_p) - K,\ K_m = B_p^+ B_m \quad (7.2.24)$$

显然若条件式 (7.2.19) 成立则可以得出 $\dot{e} = (A_p - B_pK)e$。

- **参考模型阶次不等于系统阶次**：引入辅助模型

$$\dot{x}_a = A_a x_a + B_a u_a,\ y_a = C_a u_a \quad (7.2.25)$$

其中 A_a 和 B_a 分别定义为

$$A_a = A_p + B_p(C_pB_p)^+(A_mC_p - C_pA_p),\quad B_a = B_p(C_pB_p)^+B_m \quad (7.2.26)$$

这样就可以定义 $e = x_a - x_p$, 使得 $\dot{e} = A_pe + (A_a - A_p)x_p + B_au_m - B_pu_p$, 可以选择控制律

$$u_p = Ke + [(C_pB_p)^+A_mC_p - C_pA_p)x_a + (C_pB_p)^+B_mu_m \quad (7.2.27)$$

很显然若条件式 (7.2.22) 成立，则有 $\dot{e} = (A_p - B_pK)e$。

在下一节中还将介绍基于二次型指标的最优模型跟踪控制问题。

7.3 线性二次型最优控制器设计

7.3.1 线性二次型指标与 Riccati 方程求解

假设线性 LTI 系统的状态方程模型为

$$\dot{x}(t) = Ax(t) + Bu(t),\quad y(t) = Cx(t) + Du(t) \quad (7.3.1)$$

如果想使这样一个系统能够满足某种最优的要求，最简单地可以引入线性二次型 (linear quadratic，简称 LQ) 最优控制指标[3]，亦即定义下面的目标函数

$$J = \frac{1}{2}x^T(t_f)Sx(t_f) + \frac{1}{2}\int_{t_0}^{t_f}\left[x^T(t)Q(t)x(t) + u^T(t)R(t)u(t)\right]\mathrm{d}t \quad (7.3.2)$$

其中 $Q(t)$ 和 $R(t)$ 分别是对状态变量和输入向量的加权矩阵，t_f 为终止时间。一般情况下，通常假定这两个矩阵为定常矩阵，并简记为 $Q = Q(t)$ 与 $R = R(t)$。如果想使得目标函数取得最小值，则可以首先构造一个 Hamilton 函数

$$H = -\frac{1}{2}\left[x^T(t)Qx(t) + u^T(t)Ru(t)\right] + \lambda^T(t)\left[Ax(t) + Bu(t)\right] \quad (7.3.3)$$

当输入信号不受约束时，则可以对 Hamilton 函数进行求导并令其值为 0 来求出最小值

$$\frac{\partial H}{\partial u} = -Ru(t) + B^T\lambda(t) = 0 \quad (7.3.4)$$

从而得出最优控制信号 $u^*(t)$ 满足

$$u^*(t) = R^{-1}B^T\lambda(t) \tag{7.3.5}$$

另外可以证明，$\lambda(t)$ 矩阵可以由下面的式子求出 $\lambda(t) = -P(t)x(t)$，这样可以将最优控制函数改写成

$$u^*(t) = -R^{-1}B^TP(t)x(t) \tag{7.3.6}$$

且 $P(t)$ 矩阵可以由著名的微分型 Riccati 方程求出

$$\dot{P}(t) = -P(t)A - A^TP(t) + P(t)BR^{-1}B^TP(t) - Q \tag{7.3.7}$$

可以看出，上面的微分型 Riccati 方程求解起来是很困难的，且 $u^*(t)$ 不易实现，所以我们往往可以求出其稳态解。例如前面目标函数中指定的终止时间 t_f 可以设置成 $t_f \to \infty$，这样为保证系统状态渐近地趋于零值，可以得出矩阵 $P(t)$ 趋近于一个常值矩阵，且 $\dot{P}(t) = 0$，这样式 (7.3.7) 中给出的微分型 Riccati 方程就简化为

$$-PA - A^TP + PBR^{-1}B^TP - Q = 0 \tag{7.3.8}$$

上面给出的方程又称为代数 Riccati 方程。代数 Riccati 方程的求解就要简单得多了，因为它的求解只涉及矩阵运算，所以很适合用 MATLAB 来处理。

当然求解代数 Riccati 方程的算法是各种各样的，这里只介绍一种最简单的也是最不可靠的迭代方法来求解该方程，令 $\Phi_0 = 0$，则可以写出下面的迭代公式

$$\Phi_{i+1} = E^T\Phi_iE - \left(E^T\Phi_iG + W\right)\left(G^T\Phi_iG + H\right)^{-1}\left(E^T\Phi_iG + W\right) + Q \tag{7.3.9}$$

式中

$$\begin{aligned} E &= (I-A)^{-1}(I+A), \quad G = 2(I-A)^{-2}B \\ H &= R + B^T(I-A^T)^{-1}Q(I-A)^{-1}B, \quad W = Q(I-A)^{-1}B \end{aligned} \tag{7.3.10}$$

如果 Φ_{i+1} 收敛于一个常数矩阵，即 $\|\Phi_{i+1} - \Phi_i\| < \varepsilon$，则可以得出代数 Riccati 方程的解为

$$P = 2(I - A^T)^{-1}\Phi_{i+1}(I - A)^{-1} \tag{7.3.11}$$

上面的迭代算法可以容易地用 MATLAB 语言编程实现：

```
I=eye(size(A)); iA = inv(I-A);
E = iA*(I+A);
G = 2*iA^2*B;
H = R+B'*iA'*Q*iA*B;
W = Q*iA*B;
P0 = zeros(size(A)); i=0;
while (1), i=i+1;
   P = E'*P0*E-(E'*P0*G+W)*inv(G'*P0*G+H)*(E'*P0*G+W)'+Q;
   if (norm(P-P0)<eps), break; else, P0=P; end
end
P = 2*iA'*P*iA;
```

在这里我们假设上面的程序段可以存储到文件 mylq.m 中，若给出了 A,B,Q,R 各个矩阵的值后，则系统的 Riccati 方程的解可以调用此程序来求取。

例 7.5 考虑下面给出的状态方程模型

$$\dot{x}(t) = \begin{bmatrix} -0.2 & 0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 1.6 & 0 & 0 \\ 0 & 0 & -14.3 & 85.8 & 0 \\ 0 & 0 & 0 & -33.3 & 100 \\ 0 & 0 & 0 & 0 & -10 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} u(t)$$

选择 $Q = \mathrm{diag}\{1,0,0,0,0\}$ 且 $R = 1$, 则由下面的命令可以直接求出系统的 Riccati 方程解

```
>> A=[-0.2,0.5,0,0,0;0,-0.5,1.6,0,0;0,0,-14.3,85.8,0;0,0,0,-33.3,100;0,0,0,0,-10];
>> B=[0; 0; 0; 0; 30]; Q=diag([1,0,0,0,0]); R=1;
>> mylq
>> R
R =   0.3563    0.0326    0.0026    0.0056    0.0309
      0.0326    0.0044    0.0004    0.0009    0.0056
      0.0026    0.0004    0.0000    0.0001    0.0005
      0.0056    0.0009    0.0001    0.0002    0.0012
      0.0309    0.0056    0.0005    0.0012    0.0088
```

可以查询出来，总共的迭代步数为 257，可见用这样的迭代方法并不能有效地求解高阶系统的代数 Riccati 方程。通过得出的 Riccati 方程解矩阵 P, 可以立即求出反馈矩阵 K

```
>> K=inv(R)*B'*P
K =  0.9260    0.1678    0.0157    0.0371    0.2653
```

事实上， MATLAB 的控制系统工具箱中也提供了求解代数 Riccati 方程的函数 lqr(), 其调用格式为

```
[K, P, E]=lqr(A, B, Q, R)
```

其中输入矩阵 A，B，Q，R 的意义是相当明显的，返回的 K 矩阵为状态反馈矩阵， P 为 Riccati 方程的解，而 E 为闭环系统的极点。这里的求解是建立在 MATLAB 的控制系统工具箱中给出的一个基于 Schur 变换的 Riccati 方程求解函数 are() 的基础上的，该函数的调用格式为

```
X = are(M, T, V)
```

其中 M,T,V 矩阵满足下列的代数 Riccati 方程

$$MX + XM^T - XTX + V = 0 \tag{7.3.12}$$

和前面给出的代数 Riccati 方程比较一下就很容易得出

$$M = -A,\ \ T = BR^{-1}B^T,\ \ V = -Q \tag{7.3.13}$$

对前面的例子来说，直接调用 lqr() 函数则可以得出如下结果

```
>>  [s,k,r]=lqr(A,B,Q,R)
s =    0.9260    0.1678    0.0157    0.0371    0.2653
k =    0.3563    0.0326    0.0026    0.0056    0.0309
       0.0326    0.0044    0.0004    0.0009    0.0056
       0.0026    0.0004    0.0000    0.0001    0.0005
       0.0056    0.0009    0.0001    0.0002    0.0012
       0.0309    0.0056    0.0005    0.0012    0.0088
r =  -33.3006
     -13.7976
     -11.5950
      -3.7825 + 4.9341i
      -3.7825 - 4.9341i
```

可以看出采用这一函数的求解速度明显地快于前面的迭代算法，所以在设计 LQ 最优控制器时最好能够采用这样的有效算法来完成。

例 7.6 再考虑下面的原始状态方程模型

$$\dot{x}(t)=\begin{bmatrix}-2&-6&3&-7&6\\0&-5&4&-4&8\\0&2&0&2&-2\\0&6&-3&5&-6\\0&-2&2&-2&5\end{bmatrix}x(t)+\begin{bmatrix}-2&7\\-8&-5\\-3&0\\1&5\\-8&0\end{bmatrix}u(t)$$

如果将加权矩阵 Q 和 R 分别选择为 $Q=\mathrm{diag}\{1,2,3,4,5\}$，$R=1$，则调用上面给出的 MATLAB 程序可以求解相应的代数 Riccati 方程

```
>>  A=[-2,-6,3,-7,6; 0,-5,4,-4,8; 0,2,0,2,-2; 0,6,-3,5,-6; 0,-2,2,-2,5];
>>  B=[-2,7; -8,-5; -3,0; 1,5; -8,0];
>>  Q=diag([1,2,3,4,5]); R=eye(2);
>>  eig(A)
ans = -2.0000
      -1.0000
       3.0000
       1.0000
       2.0000
>>  mylq
??? Error using ==> norm
NaN and Inf not allowed.
```

可见原始系统是不稳定的，对此不稳定系统来说前面的算法则将出现迭代过程不收敛的现象，这样一般不能由此方法求出 Riccati 方程的解。这时若调用 MATLAB 控制系统工具箱中提供的 lqr() 函数，则可以直接求出下面的结果

```
>>  [k,s,e]=lqr(A,B,Q,R)
k =    0.0609  -92.7396   22.7997  -92.4740   68.6507
       0.2598   15.0943   -4.5399   17.2924  -11.5145
s =    1.0e+003 *
       0.0002   -0.0013    0.0007   -0.0015    0.0007
```

```
  -0.0013    2.0673   -0.4666    2.0721   -1.6214
   0.0007   -0.4666    0.1103   -0.4685    0.3636
  -0.0015    2.0721   -0.4685    2.0777   -1.6247
   0.0007   -1.6214    0.3636   -1.6247    1.2732
e = -22.5972
    -13.4340
     -2.1474 + 0.6185i
     -2.1474 - 0.6185i
     -1.1990
```

从上面的结论可见，尽管原始对象模型为不稳定的，经过这里的 LQ 最优状态反馈之后，将得出闭环稳定的系统，这样我们说 LQ 控制策略可以对原始系统进行镇定。系统的各个状态曲线可以由下面的 MATLAB 命令仿真出来，并在图 7-5 中给出。

```
>> t=0:.1:10; C=eye(5); D=zeros(5,2);
>> y1=step(A-B*k,B,C,D,1,t);
>> y2=step(A-B*k,B,C,D,2,t);
>> plot(t,y1,t,y2)
```

可见最优控制之后系统的各个状态都是稳定的。

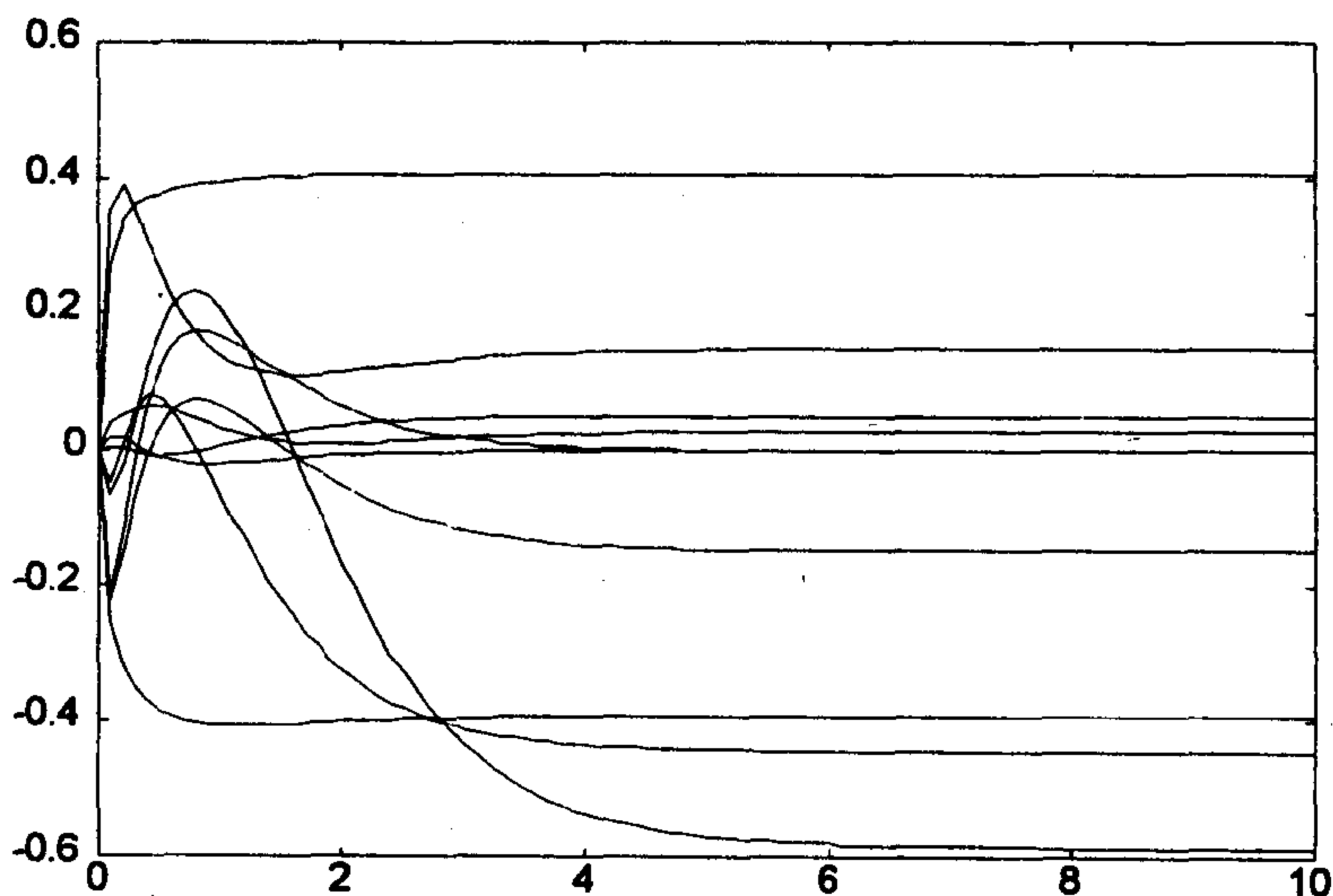


图 7-5　最优控制状态变量曲线

通常系统的状态信号并不是可以直接测取到的，所以往往需要为系统构造一个状态观测器，由该观测器来重构不可测的状态变量，并将之用于 LQ 型的最优控制中。这样的控制系统框图结构如图 7-6 所示，状态观测器的种类很多，在这里就不再进行介绍了。

值得指出的是，这样得出的最优控制是“人工”意义下的最优控制，因为它的效果将完全取决于加权矩阵 Q 和 R 的选取，如果这些加权矩阵选择不当，则可能得出完全没有意义的解，更谈不上“最优”了。



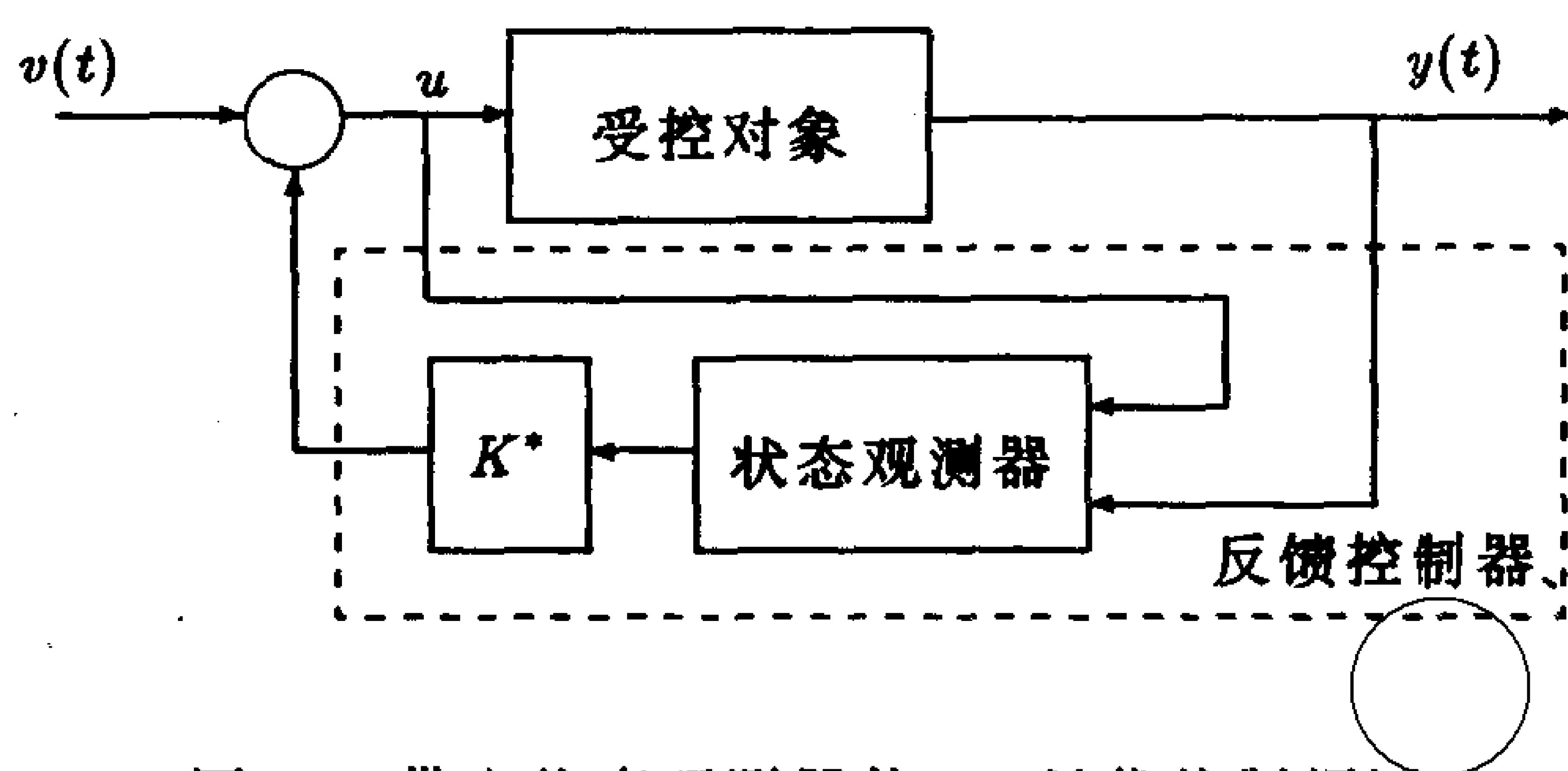


图7-6 带有状态观测器的 LQ 最优控制框图

7.3.2 输出反馈的线性二次型最优控制

在很多情况下用户想采用输出量而不是状态变量或其观测值来作二次型指标最优控制，这样就需要对前面的 LQ 算法进行修正，假设引入输出最优反馈控制

$$u(t) = -K_o y(t) = -K_o C x(t) \tag{7.3.14}$$

则最优反馈矩阵 K_o 应该使得 $A - BK_oC$ 为渐近稳定矩阵，且 K_o 可以如下求得 [21]

$$K_o = R^{-1}B^T PZC^T(CZC^T)^{-1} \tag{7.3.15}$$

其中 P, Z 矩阵分别满足

$$P(A - BK_oC) + (A - BK_oC)^T P + C^T K_o^T RK_oC + Q = 0 \tag{7.3.16}$$

$$Z(A - BK_oC)^T + (A - BK_oC)Z + I_n = 0 \tag{7.3.17}$$

显然要求解 K_o 需要涉及迭代过程，可以根据上面算法由 MATLAB 编写出下面的函数来求取基于输出的二次型最优控制输出反馈矩阵 K_o

```
function Ko=outlqr(A,B,C,Q,R,K,tol)
if (nargin==6), tol=1e-10; end
K1=K; I=eye(size(A));
while (1)
   A0=A-B*K1*C;
   P=lyap(A0',C'*K1'*R*K1*C+Q);
   Z=lyap(A0,I);
   Ko=inv(R)*B'*P*Z*C'*inv(C*Z*C');
   if (norm(Ko-K1, 1)>tol), K1=Ko; else, break; end
end
```

在这里带有了收敛条件为 $\|K_o - K_1\| < \text{tol}$, 并将其默认值置为 $\text{tol} = 10^{-10}$, 调用此函数应该首先给出输出反馈矩阵的初值 K_1, 然后由此值出发进行迭代，最终求出系统的最优反馈矩阵 K_o 。注意，由此迭代过程得出的输出反馈矩阵不能保证满足使得闭环系统稳定的前提条件，所以应该在得出矩阵后测试一下系统的闭环稳定性。

例 7.7 考虑例 7.5 系统，若输出方程可以写成 $y = [1,0,4,3,2]x(t)$，则可以由下面 MATLAB 语句求出最优输出反馈矩阵 K_o 及闭环极点

```
>> A=[-0.2,0.5,0,0,0;0,-0.5,1.6,0,0;0,0,-14.3,85.8,0;0,0,0,-33.3,100;0,0,0,0,-10];
>> B=[0; 0; 0; 0; 30]; Q=diag([1,0,0,0,0]); R=1;
>> C=[1,0,4,3,2];
>> Ko=outlqr(A,B,C,Q,R,1)
Ko =   1.5307
>> eig(A-B*Ko*C)
ans = 1.0e+002 *
      -0.1561 + 1.2314i
      -0.1561 - 1.2314i
      -1.1821
      -0.0035 + 0.0039i
      -0.0035 - 0.0039I
>> t=0:0.001:.2;
>> y=step(A-B*Ko*C,B,C,0,1,t); plot(t,y)
```

闭环系统的阶跃响应曲线如图 7-7 所示。用户还可以和校正前的系统进行比较

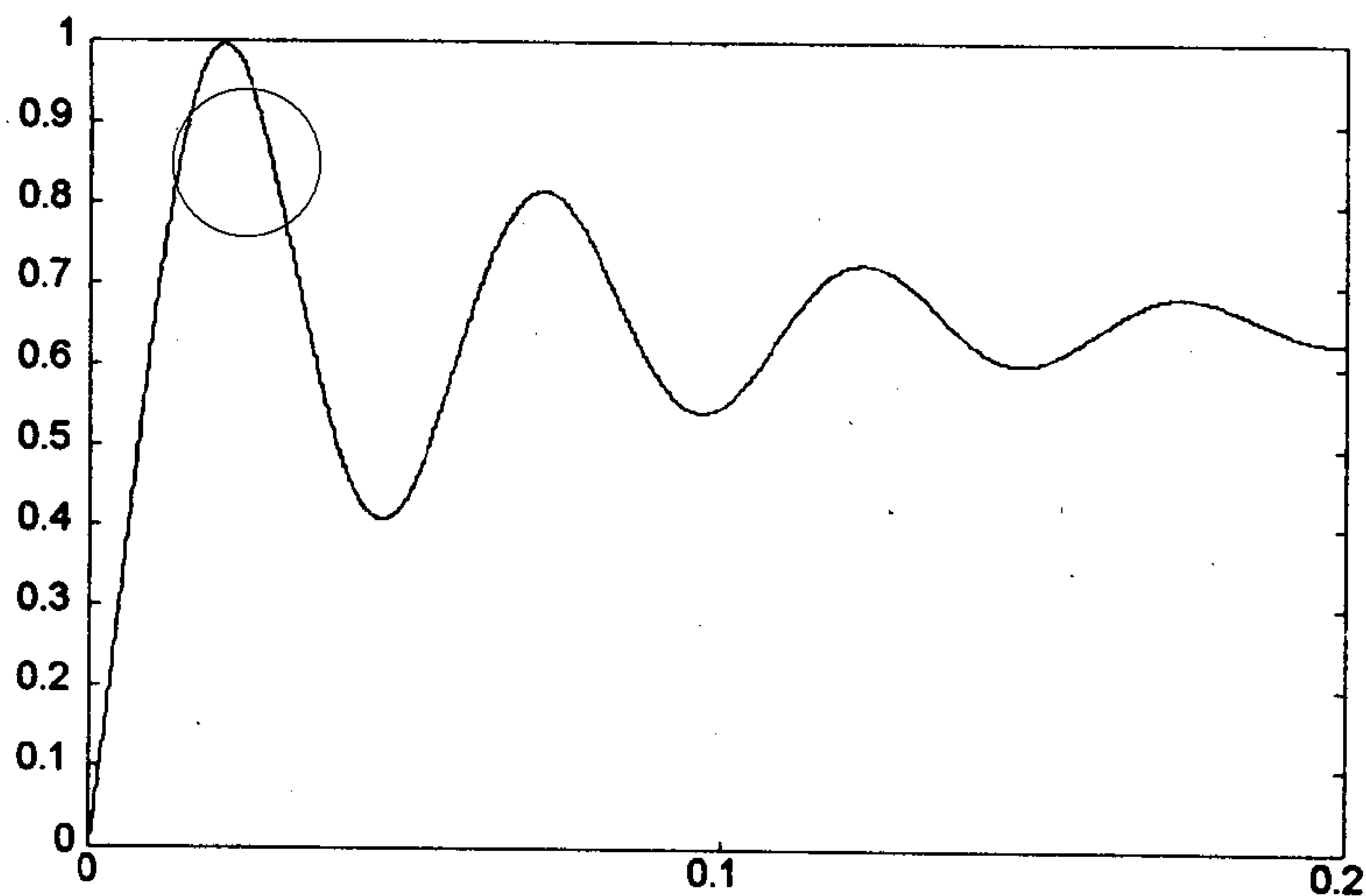


图 7-7 最优控制输出曲线

```
>> t=0:.1:20;
>> y=step(A,B,C,0,1,t); plot(t,y)
```

原系统的阶跃响应如图 7-8 所示，可见最优控制施加之后该系统的响应有了明显的改善。用户还可以通过调节 Q 和 R 加权矩阵的方法进一步改善系统的输出响应。



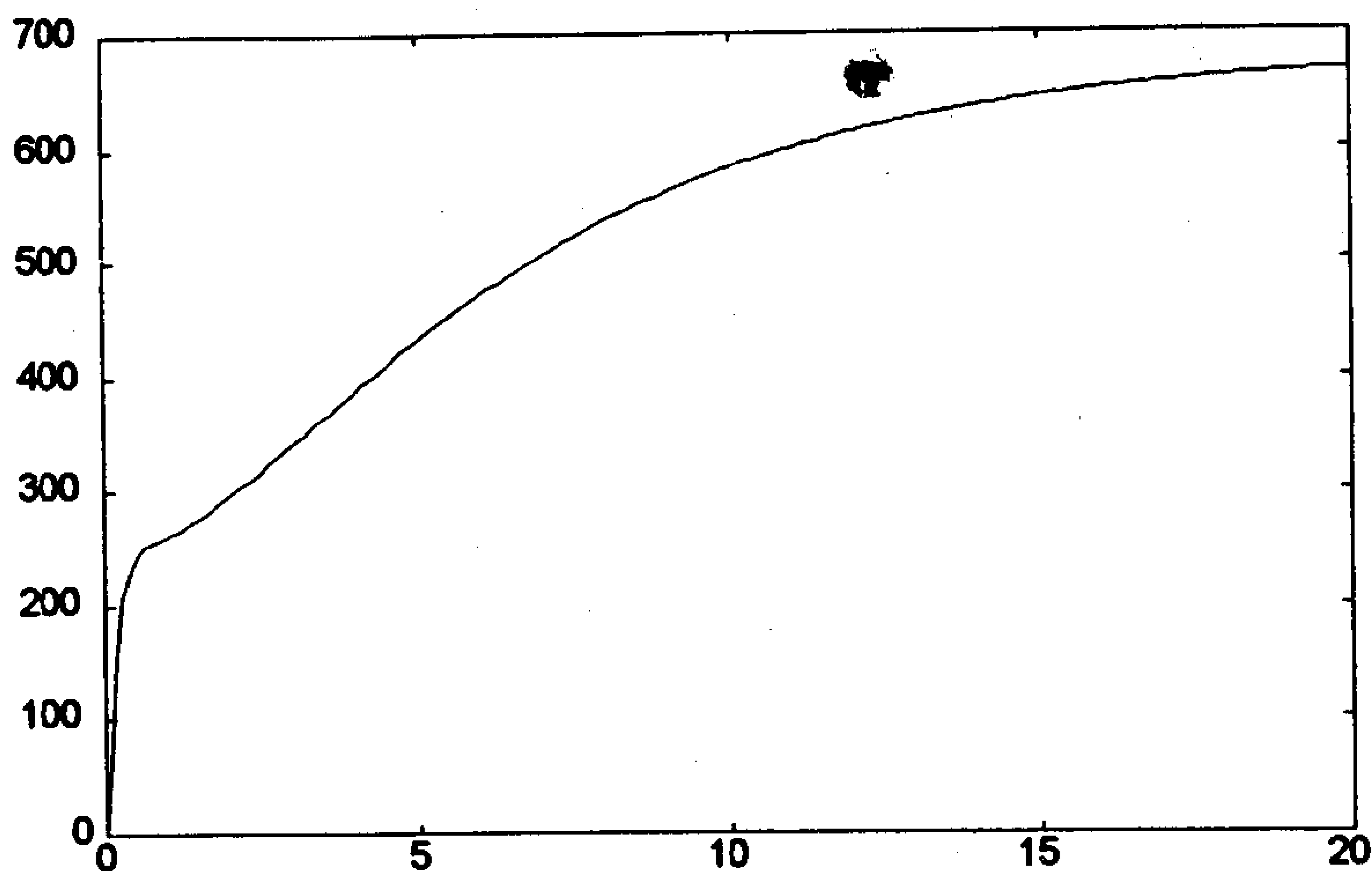


图7-8 原系统的输出阶跃响应曲线

7.3.3 最优模型跟踪问题

前面介绍过模型跟踪控制的基本概念，若假定系统模型和要跟踪的模型的状态方程为

$$\dot{x} = A_p x_p + B_p u_p, \quad y_p = C_p x_p, \quad \dot{x}_m = A_m x_m \tag{7.3.18}$$

其中 x_m 为 m 列向量，这时的目的是找出一个控制律 u_p 使得 $y_p \to x_m$，这可以定义出下面的最优化指标

$$J = \int_0^{\infty} \left[(\dot{y}_p - A_m y_p)^T Q (\dot{y}_p - A_m y_p) + u_p^T R u_p \right] \mathrm{d}t \tag{7.3.19}$$

其中 R 为正定矩阵，且 Q 为非负定矩阵。若系统对象模型为可控的，且对象模型为稳定的，则此性能指标是有界的，可以证明这时最优控制律可以写成

$$u_p = -R_1^{-1} \left[B_p^T Q (C_P A_P - A_m C_p) + B_p^T P \right] x_p \tag{7.3.20}$$

式中 P 为下面 Riccati 方程的对称正定解

$$P A_1 + A_1^T P - P B_p R_1^{-1} B_p^T P + C_1^T Q_1 C_1 = 0 \tag{7.3.21}$$

其中 $R_1 = B_p^T C_p^T Q C_p B_p + R$, $A_1 = A_p - B_p R^{-1} B_p^T C_p Q (C_p A_p - A_m C_p)$, $Q_1 = Q - Q C_p B_p R_1^{-1} B_p^T C_p^T Q$, $C_1 = C_p A_p - A_m C_p$ 。

例 7.8 考虑下面的一个 3 输入系统模型

$$A_p = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1.4 \times 10^{-4} & -2.04 & -1.95 & 0.0133 \\ -2.5 \times 10^{-4} & 1 & -1.32 & -0.0238 \\ -0.56 & 0 & 0.36 & -0.028 \end{bmatrix}, \quad B_p = \begin{bmatrix} 0 & 0 & 0 \\ -5.33 & 6.45 \times 10^{-3} & -0.267 \\ -0.16 & -1.155 \times 10^{-2} & -0.251 \\ 0 & 0.106 & 0.0862 \end{bmatrix}$$

若想对系统的各个状态变量进行模型跟踪控制，则可以选择输出矩阵 $C_p=I_4$，并如下选择下面的 4 阶参考模型

$$A_m=\begin{bmatrix}0 & 1 & 0 & 0\\ 5.3\times 10^{-7} & -0.418 & -0.1202 & 2.32\times 10^{-3}\\ -4.6\times 10^{-9} & 1 & -0.75 & -2.39\times 10^{-2}\\ -0.56 & 0 & 0.3 & -1.743\times 10^{-2}\end{bmatrix},\ B_m=\begin{bmatrix}0 & 0\\ -0.172 & 7.56\times 10^{-6}\\ -0.024 & -7.78\times 10^{-5}\\ 0 & 3.68\times 10^{-3}\end{bmatrix}$$

选择加权矩阵 $Q=\mathrm{diag}(0.1,1,0.1,0), R=I_3$，则由下面的 MATLAB 命令可以得出系统的最优控制

```
>> Ap=[0,1,0,0; 1.4e-4,-2.04,-1.95,0.0133;
      -2.5e-4,1,-1.32,-0.0238; -0.56,0,0.36,-0.028];
>> Bp=[0,0,0; -5.33,6.45e-3,-0.267; -0.16,-1.155e-2,-0.251; 0,0.106,0.0862];
>> Am=[0,1,0,0; 5.3e-7,-0.418,-0.1202,2.32e-3;
      -4.6e-9,1,-0.75,-2.39e-2; -0.56,0,0.3,-1.743e-2];
>> Bm=[0,0; -0.172,7.56e-6; -0.024,-7.78e-5; 0,3.68e-3]; Cp=eye(size(Ap));
>> Q=diag([0.1,1,0.1,0]); R=eye(3);
>> R1=Bp'*Cp'*Q*Cp*Bp+R;  A1=Ap-Bp*inv(R1)*Bp'*Cp*Q*(Cp*Ap-Am*Cp);
>> Q1=Q-Q*Cp*Bp*inv(R1)*Bp'*Cp'*Q;  C1=Cp*Ap-Am*Cp;
>> P=are(A1,Bp*inv(R1)*Bp',C1'*Q1*C1);
>> K=-inv(R1)*(Bp'*Q*(Cp*Ap-Am*Cp)+Bp'*P)
K =  0.0017    -0.2580    -0.3207    0.0014
     0.0000     0.0013     0.0004    0.0000
    -0.0003     0.0019    -0.0173   -0.0001
```

7.4 线性二次型 Gauss 最优控制问题

下面我们将讨论一种更复杂的情况，当状态变量量测时如果存在随机扰动将如何进行最优控制设计，这涉及到线性二次型 Gauss (linear quadratic Gaussian, 简称 LQG) 问题的求解，在本节中将介绍 LQG 设计方法。

7.4.1 LQG 问题的一般解法

假设受控对象的状态方程模型可以写成

$$\dot{x}(t)=Ax(t)+Bu(t)+\Gamma w(t),\quad y(t)=Cx(t)+v(t) \tag{7.4.1}$$

式中 $w(t)$ 和 $v(t)$ 均为白噪声信号，分别是对状态变量量测和输出变量量测的随机扰动，它们的协方差矩阵分别为

$$E[w(t)w^T(t)]=W\geq 0,\quad E[v(t)v^T(t)]=V>0 \tag{7.4.2}$$

且为零均值 Gauss 随机过程，式中 $E[x]$ 表示求取 x 向量的均值，而 $E[xx^T]$ 表示零均值向量 x 的协方差矩阵。此外 $w(t)$ 和 $v(t)$ 为互不相关的信号，亦即 $E[w(t)v^T(t)]=0$。类似于前面叙述的 LQ 问题，系统的性能指标定义为

$$J=E\left\{\int_0^\infty\left[z^T(t)Qz(t)+u^T(t)Ru(t)\right]\mathrm{d}t\right\} \tag{7.4.3}$$

式中 $z(t)=Mx(t)$ 为状态变量 $x(t)$ 的某种线性组合，加权矩阵 Q 和 R 分别为对称半正定矩阵和对称正定矩阵，亦即 $Q=Q^T\geq 0$，$R=R^T>0$，这样 LQG 问题可以分解为下面两个子问题来进行求解，首先获得一个使 $E\Big\{[x(t)-\hat{x}(t)]^T[x(t)-\hat{x}(t)]\Big\}$ 极小化的最优估计信号 $\hat{x}(t)$，然后将最优估计信号 $\hat{x}(t)$ 假设为系统的状态变量 $x(t)$，从而对这一问题求解普通的 LQ 问题来得出系统的控制器。根据 Kalman 滤波理论，可以得出如图 7-9 所

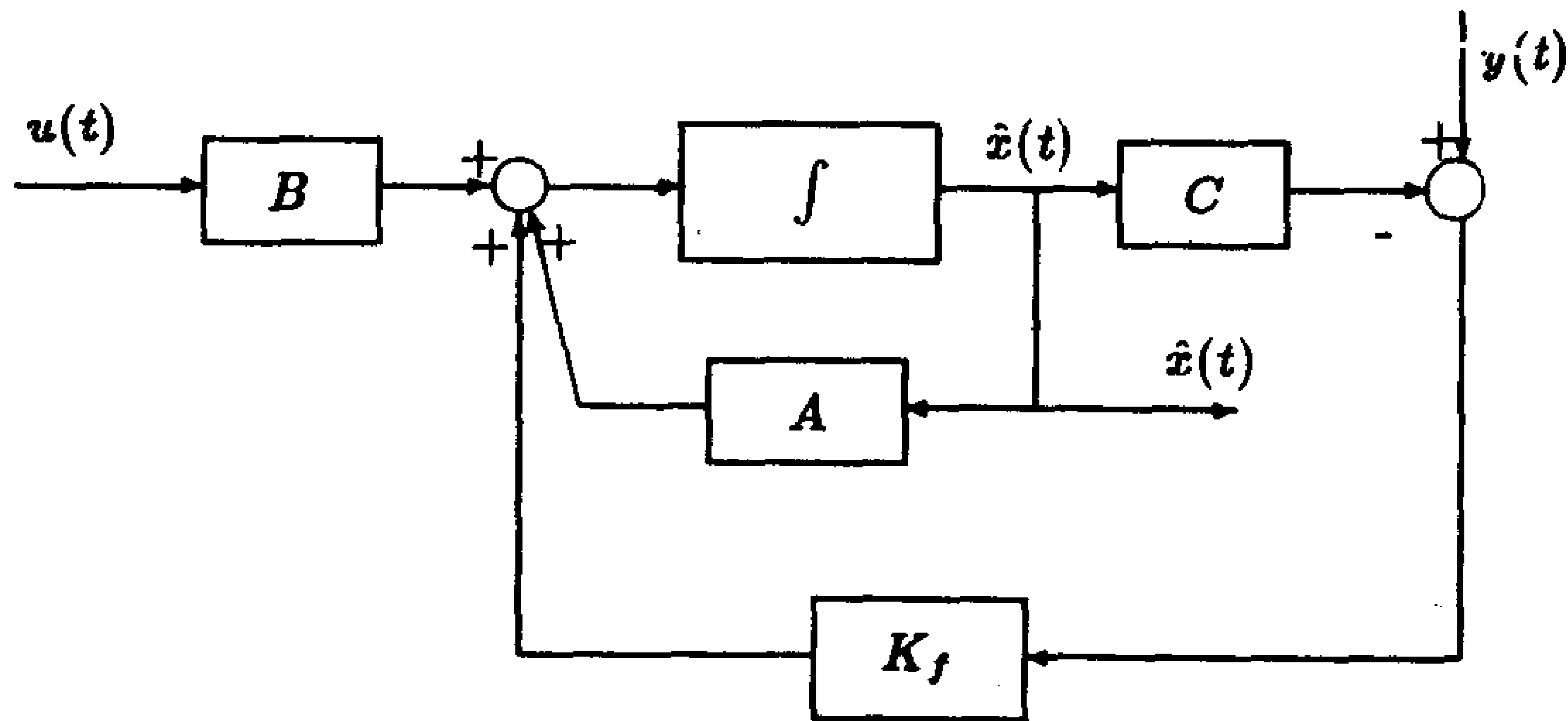


图 7-9　Kalman 滤波器框图结构

示的框图形式，这时 Kalman 滤波器的增益矩阵 K_f 可以由下式求出

$$K_f=P_fC^TV^{-1} \tag{7.4.4}$$

其中 P_f 满足下面的代数 Riccati 方程

$$P_fA^T+AP_f-P_fC^TV^{-1}CP_f+\Gamma W\Gamma^T=0 \tag{7.4.5}$$

且已知 P_f 为半正定对称矩阵，即 $P_f=P_f^T\geq 0$。得出了最优滤波信号 $\hat{x}(t)$ 之后，就可

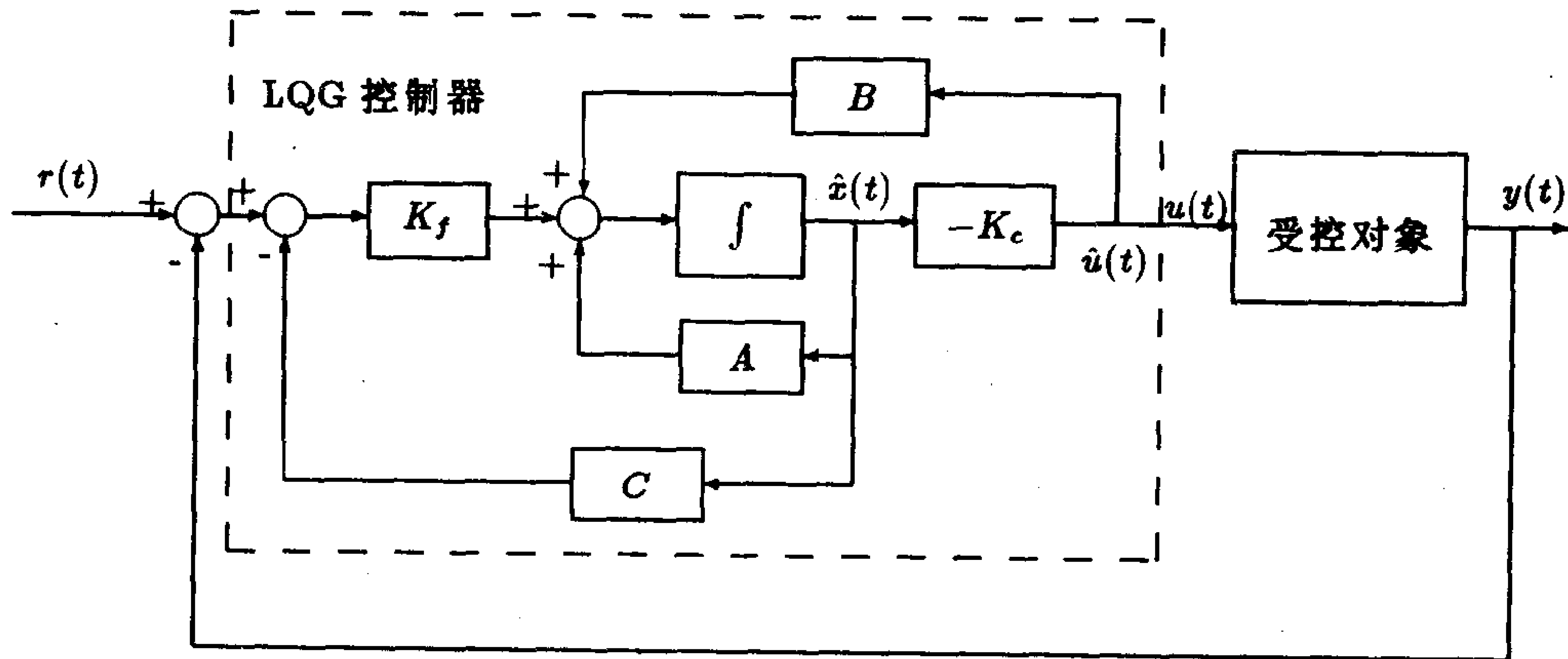


图 7-10　LQG 控制结构框图

以构造出整个 LQG 补偿器的结构图，如图 7-10 所示，最优控制向量 $u^*(t)$ 满足

$$u^*(t)=-K_c\hat{x}(t) \tag{7.4.6}$$

这里最优状态反馈矩阵 K_c 可以由下式得出

$$K_c = R^{-1}B^T P_c \tag{7.4.7}$$

而 P_c 矩阵满足代数 Riccati 方程

$$A^T P_c + P_c A - P_c B R^{-1} B^T P_c + M^T Q M = 0 \tag{7.4.8}$$

这时得出的矩阵 P_c 为半正定对称矩阵，即 $P_c = P_c^T \geq 0$ 。

7.4.2 回路传输恢复技术

由上面给出的方法可以看出，求解两个独立的 Riccati 方程就可以设计出带有 Kalman 滤波的 LQG 控制器，因为前面涉及了最优 Kalman 滤波并由最优状态反馈来构成控制结构，所以似乎由这样的设计就可以得出满意的控制效果，然而事情并非如此简单，这样设计出来的 LQG 控制器的稳定裕度是相当小的，若模型存在微小的偏差或受到微小的扰动，则总体的控制就可能出现不稳定的现象。以前解决 LQG 问题的一个显然的方法是使得滤波器的动态过程比状态反馈的动态过程快得多，而这种方法在实际中被证明是错误的，因为这非但不会增加系统的鲁棒性，而会使得在一些情况下系统的稳定裕度有较大的衰减。事实上，在实际设计中会发现图 7-10 中的 $u(t)$ 信号和 $\hat{u}(t)$ 处的返回比矩阵相差甚远，要解决 LQG 的一种有效的方法是引入回路传输恢复 (loop transfer recovery，简称 LTR) 的控制方法，使得在图 7-10 中标注为 $u(t)$ 信号处的返回比尽可能逼近 $\hat{u}(t)$ 信号处的返回比。文献 [23] 中给出了比较成型的 LTR 算法。

为解决 LTR 问题需要设计出一个很好的滤波增益 K_f，在 LTR 设计中需要假定式 (7.4.1) 中的白噪声协方差矩阵 $W = W_0 + q\Sigma$，而 W_0 为实际噪声的协方差矩阵，Σ 满足 $\Sigma = \Sigma^2 \geq 0$，且 q 理论上应该为一个任意大的数值，这时式 (7.4.5) 可以改写成

$$\frac{P_f A^T}{q} + \frac{AP_f}{q} - \frac{P_f C^T V^{-1} C P_f}{q} + \frac{\Gamma W_0 \Gamma^T}{q} + \Gamma\Sigma\Gamma^T = 0 \tag{7.4.9}$$

可以证明，若 $C(sI-A)^{-1}\Gamma W^{1/2}$ 描述的系统没有右半平面传输零点，则 $\lim_{q\to\infty} P_f/q = 0$，这样当 $q \to \infty$ 时，$K_f \to q^{1/2}\Gamma\Sigma^{1/2}V^{-1/2}$ 。特别地若选择 $\Gamma = B, \Sigma = I$，且 $C(sI-A)^{-1}B$ 在右半平面没有传输零点时，

$$K_f \to q^{1/2} B V^{-1/2}, \text{ 当 } q \to \infty \text{ 时} \tag{7.4.10}$$

对输入和输出个数相等的系统来说，可以采用下面的步骤来基于对象的输入信号来设计 LQG 控制器

- 按照线性二次型指标 Q, R 的方法单独设计最优控制器，并调整 Q 和 R 使得返回比矩阵 $-K_c(sI-A)^{-1}B$ 为令人满意的。
- 选择 $\Gamma = B, W = W_0 + qI$ 且 $V = I$，增加 q 的值，直到校正后的对象输入点处的返回比在相当大的频率范围内足够接近 $-K_c(j\omega I - A)^{-1}B$ 。注意在实际应用时不应该将 q 值选择得过大，以免产生截断误差，降低系统的鲁棒性。

基于对象输入信号的LQG控制器设计可以由鲁棒控制工具箱中的ltru()函数来完成，在实际设计中往往还需要基于对象输出信号来进行LTR设计，设计步骤为：

- 对系统进行初步的LQ二次型最优设计
- 选择一个较大的q使得可以恢复回路的传输

在鲁棒控制工具箱中基于对象输出的LTR设计可以由ltry()函数来进行，其调用格式其实还是很显然的。用户还可以参阅鲁棒控制工具箱手册[8]或联机帮助来获得必要的信息。在下面的例子中将不加格式说明地调用一些鲁棒控制工具箱中的函数来演示LTR设计。

例 7.9　假设系统的状态方程模型为

$$G(s)=\frac{-(948.12s^3+30325s^2+56482s+1215.3)}{s^6+64.554s^5+1167s^4+3728.6s^3-5495.4s^2+1102s+708.1}$$

可以如下输入系统的模型，并作初步开环分析

```
>> num=-[948.12, 30325, 56482, 1215.3];
>> den=[1, 64.554, 1167, 3728.6, -5495.4, 1102, 708.1];
>> w = logspace(-4,5,200); lenw = length(w);
>> H=freqresp(num,den,sqrt(-1)*w); H=[H;conj(H(lenw:-1:1,:))]; plot(H)
```

由上面命令可以绘制出系统的Nyquist图形，如图7-11所示，注意这里的Nyquist图形是在$\omega\in(-\infty,\infty)$

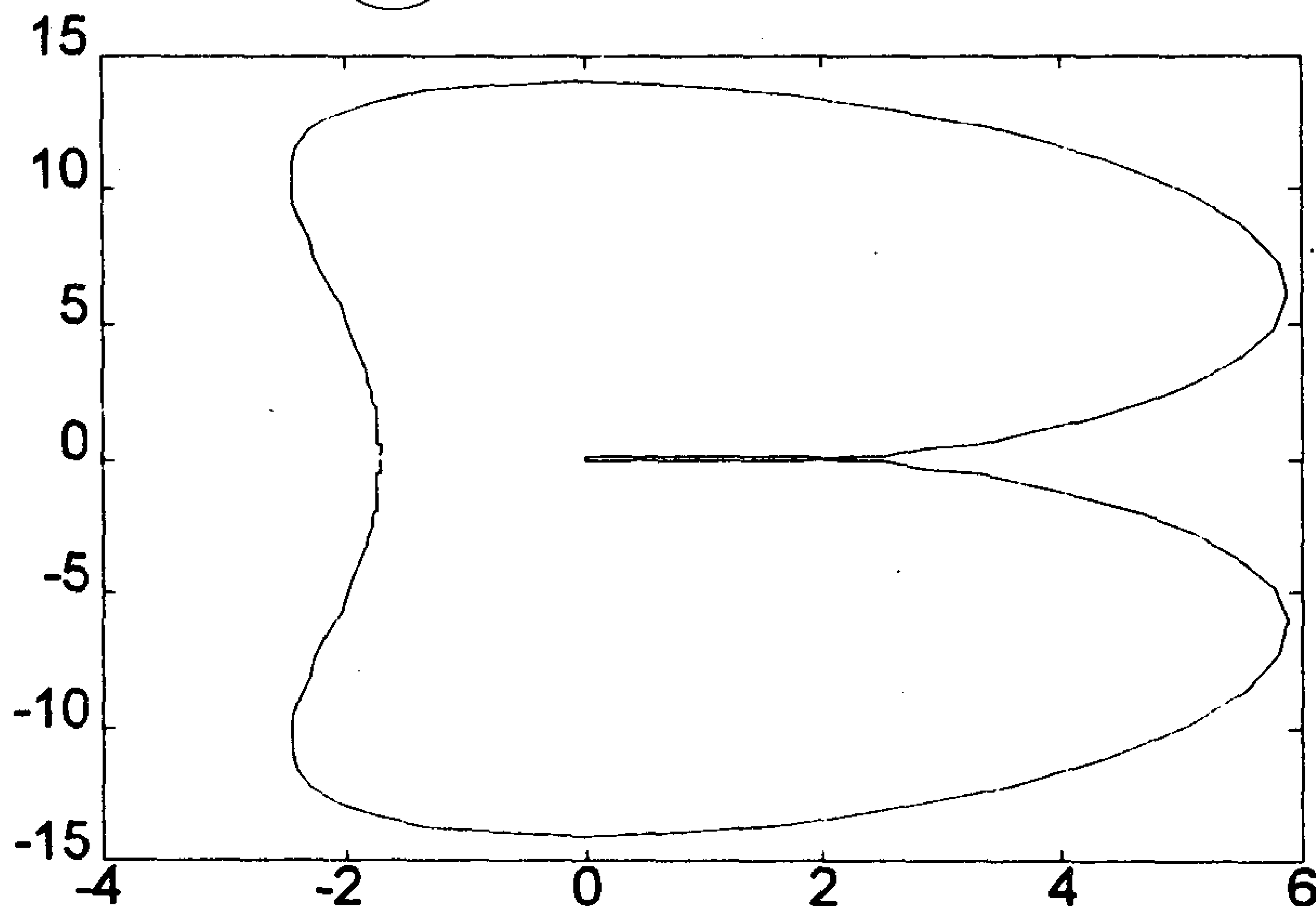


图7-11　原系统的Nyquist图形

范围内的，而不是以前介绍的$\omega\in(0,\infty)$。设加权矩阵$\Gamma=BB^T$, $V=1$则可以获得较好的Kalman滤波特性，

```
>> [a,b,c,d]=tf2ss(num,den); Gamma=b*b'; V=1; Q=c'*c; R=1;
>> Kf = lqrc(a',c',diagmx(Gamma,V))'; HH=[];
>> H1=freqresp(a,Kf,c,0,1,sqrt(-1)*w); H1=[H1; conj(H1)]; plot(H1)
```

```
>> svk=[real(H1) imag(H1)]; ss_g = mksys(a,b,c,d); q=1;
>> [ss_f,svl]=ltry(ss_g,Kf,Q,R,q,w,svk); [af,bf,cf,df]=branch(ss_f);
>> svf=sigma(af,bf,cf,df,1,w);
>> [al,bl,cl,dl]=series(af,bf,cf,df,a,b,c,d);
>> [acl,bcl,ccl,dcl]=feedbk(al,bl,cl,dl,2);
>> t=0:.05:5; y=step(acl,bcl,ccl,dcl,1,t); plot(t,y)
```

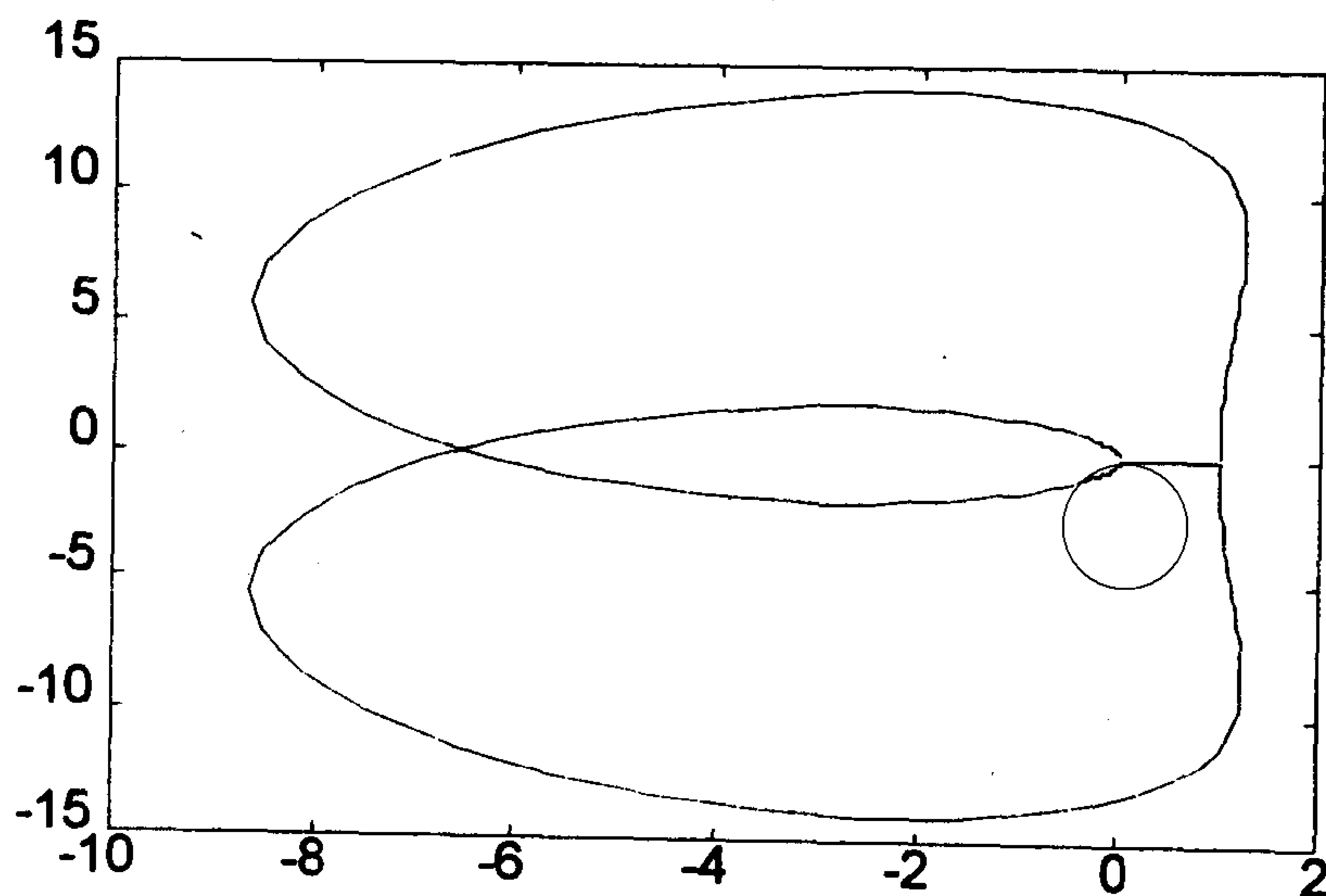


图7-12　系统的 Nyquist 曲线

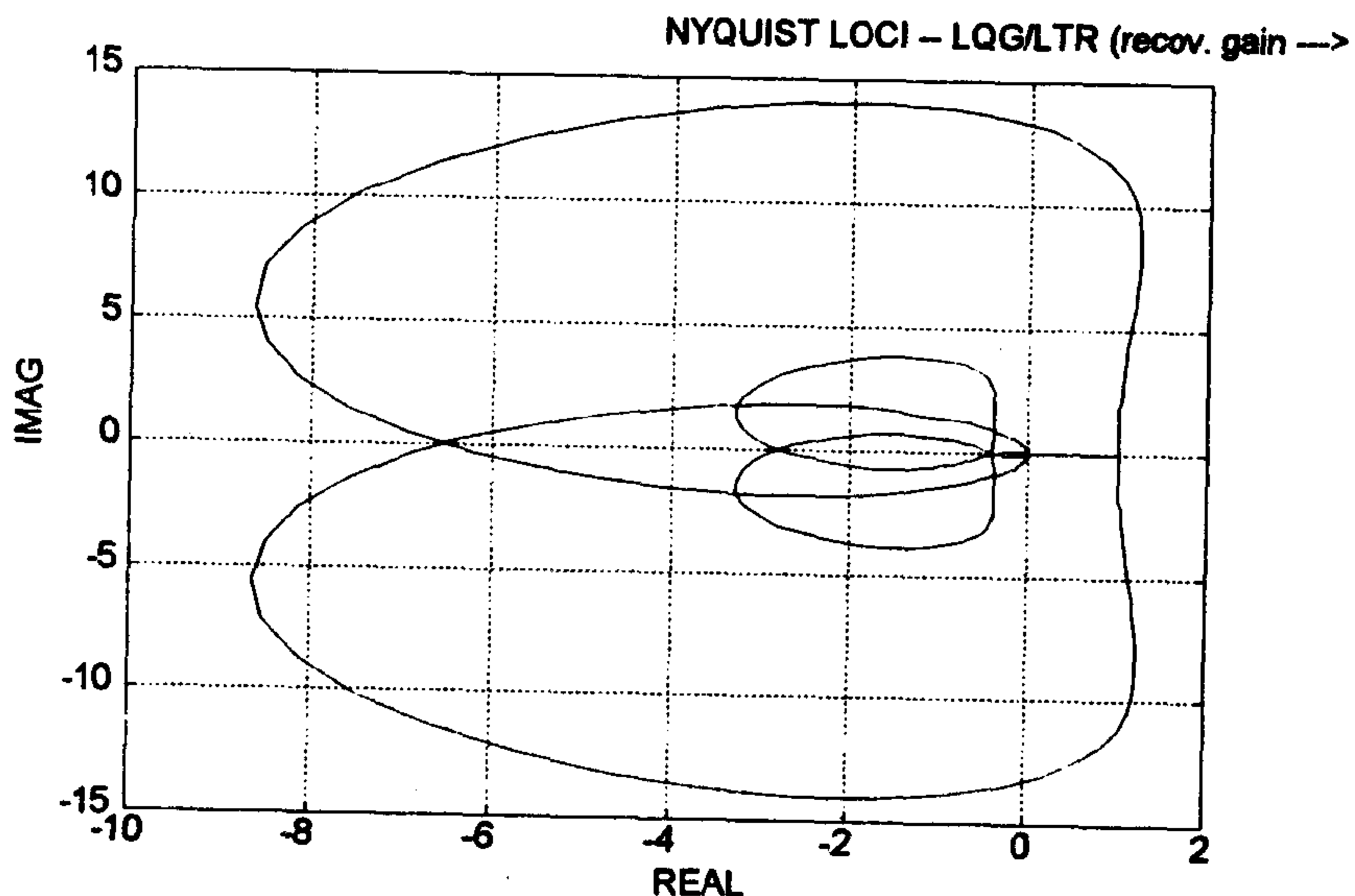


图7-13　$q=1$ 的特征轨迹曲线

由前面的程序段可以绘制出 $(a, K_f, c, 0)$ 系统的 Nyquist 曲线，如图 7-12 所示。$q=1$ 时的 $\hat{u}(t)$ 处返回比的特征轨迹曲线与 $u(t)$ 处返回比的特征轨迹曲线，如图 7-13 所示，可见它们是截然不同

的，这就是需要引入 LTR 技术的原因。其实上面的分析中还得出了系统的阶跃响应曲线，如图 7-14 所示，可见在这样的控制器下的动态响应并不是很理想。

增大 q 的值，如 $q=10^4,10^8$，则在各个 q 值处的特征轨迹曲线如图 7-15 所示，其实当 $q=10^8$ 时就几乎看不出两处的特征轨迹曲线有任何区别了，所以可以将 q 的值选作 10^8 即可。

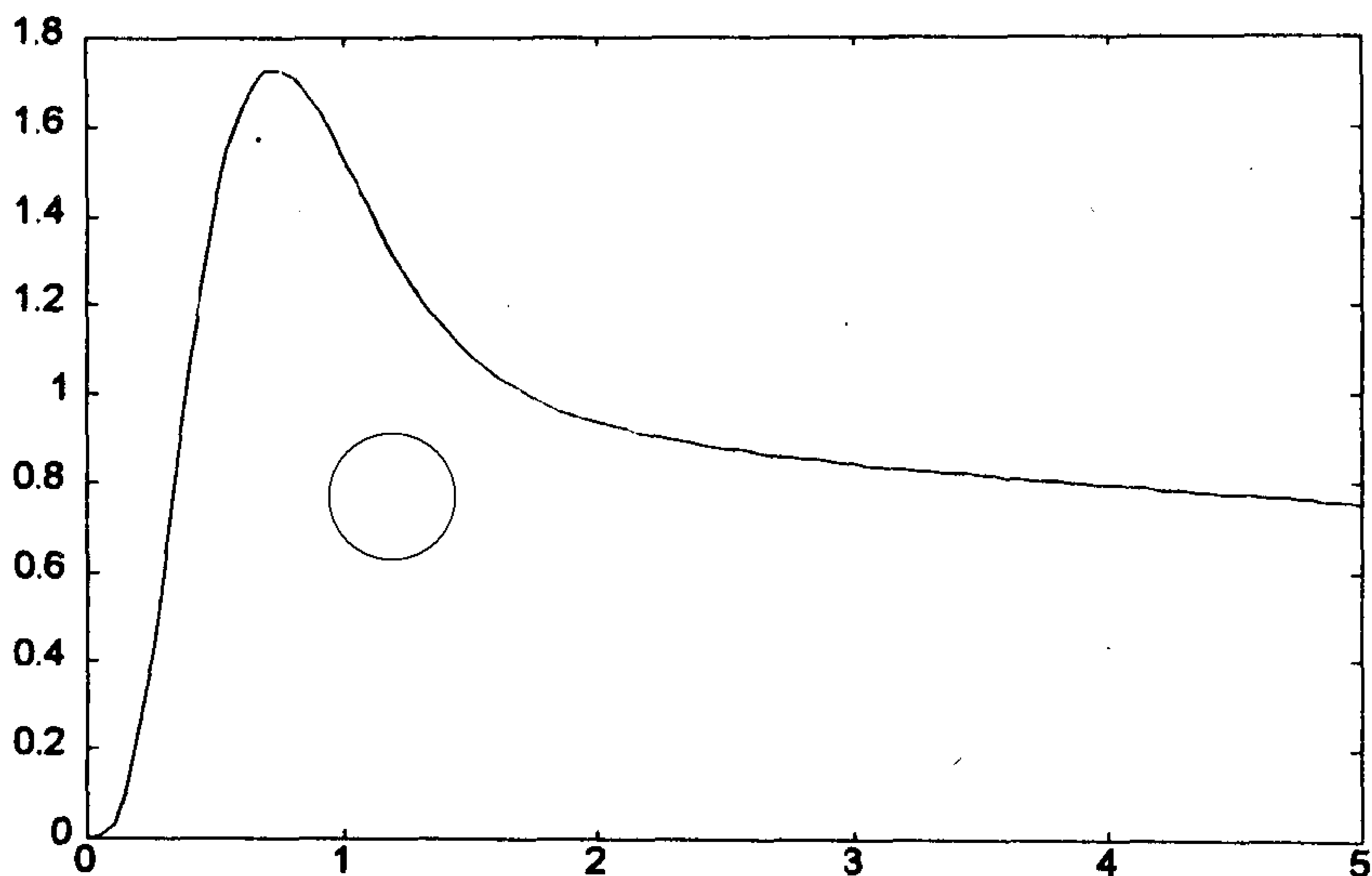


图7-14 $q=1$ 闭环系统的阶跃响应曲线

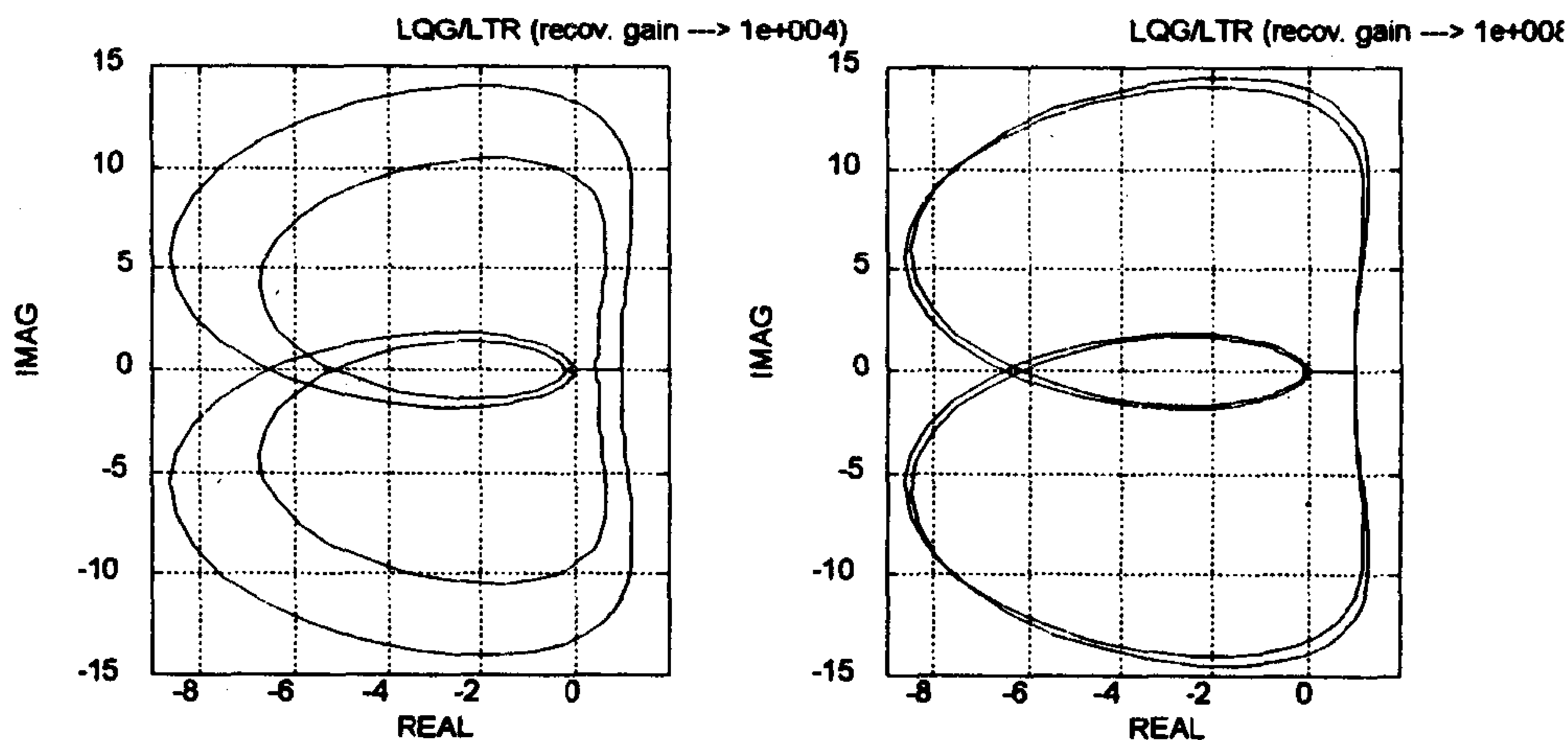


图7-15 增大 q 时的特征轨迹曲线

```
>> q=1e4; subplot(121); [ss_f,svl]=ltry(ss_g,Kf,Q,R,q,w,svk);
>> q=1e8; subplot(122); [ss_f,svl]=ltry(ss_g,Kf,Q,R,q,w,svk);
>> clg; [ss_f,svl]=ltry(ss_g,Kf,Q,R,q,w,svk);
>> [af,bf,cf,df] = branch(ss_f);
>> svf = sigma(af,bf,cf,df,1,w); svf = 20*log10(svf); semilogx(w,svf), grid
```

```
>>  [al,bl,cl,dl] = series(af,bf,cf,df,a,b,c,d);
>>  [acl,bcl,ccl,dcl] = feedbk(al,bl,cl,dl,2); polcl = eig(acl)
polcl = 1.0e+002 *
      -2.21737022829332
      -1.15194092429865 + 1.82659123700336i
      -1.15194092429865 - 1.82659123700336i
      -0.30259261916773
      -0.29759504930164
      -0.25312758011753
      -0.04686578244180 + 0.02847933756361i
      -0.04686578244180 - 0.02847933756361i
      -0.02275793583210
      -0.02064131674167
      -0.00025161080222
      -0.00021038593856
>>  t=0:.05:5; y=step(acl,bcl,ccl,dcl,1,t); plot(t,y)
```

上面的命令首先绘制出系统控制器的主导增益曲线，如图 7-16 所示， 然后调用相应的函数求出系

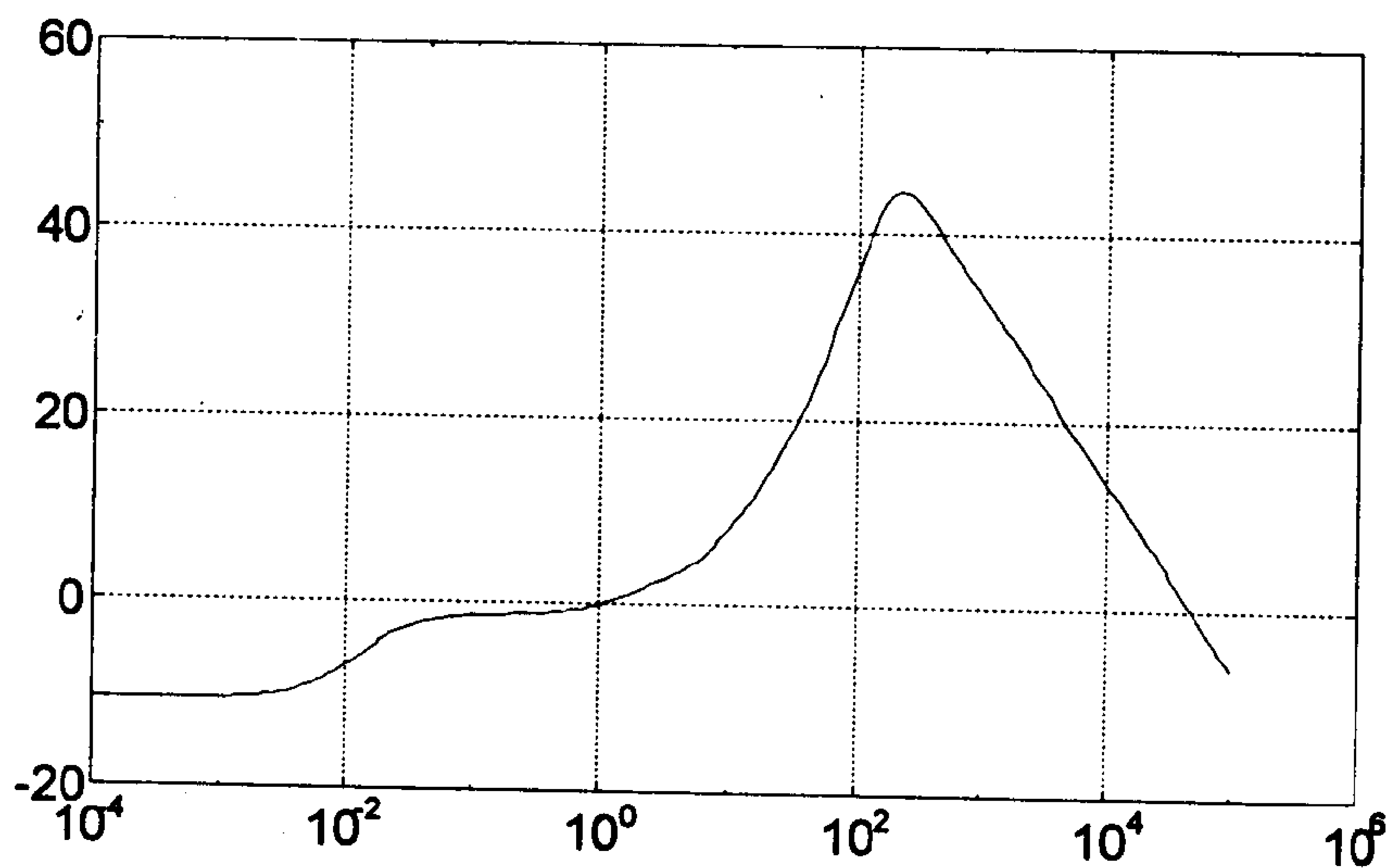


图7-16　控制器的主导增益曲线

统的闭环系统模型 acl，bcl，ccl，dcl，并求出闭环系统的所有特征值，可见闭环系统是稳定的，这时还可以绘制出系统的阶跃响应曲线，如图 7-17 所示。

7.5　基于 H_∞ 技术的鲁棒控制方案

7.5.1　范数指标与 H_∞ 空间的基本概念

单变量系统的研究中经常使用频率响应来描述系统的特性，因为该特性在每一个频率点处会取唯一的值，描述起来还是很方便和直观的。对多变量系统来说因为在每个频



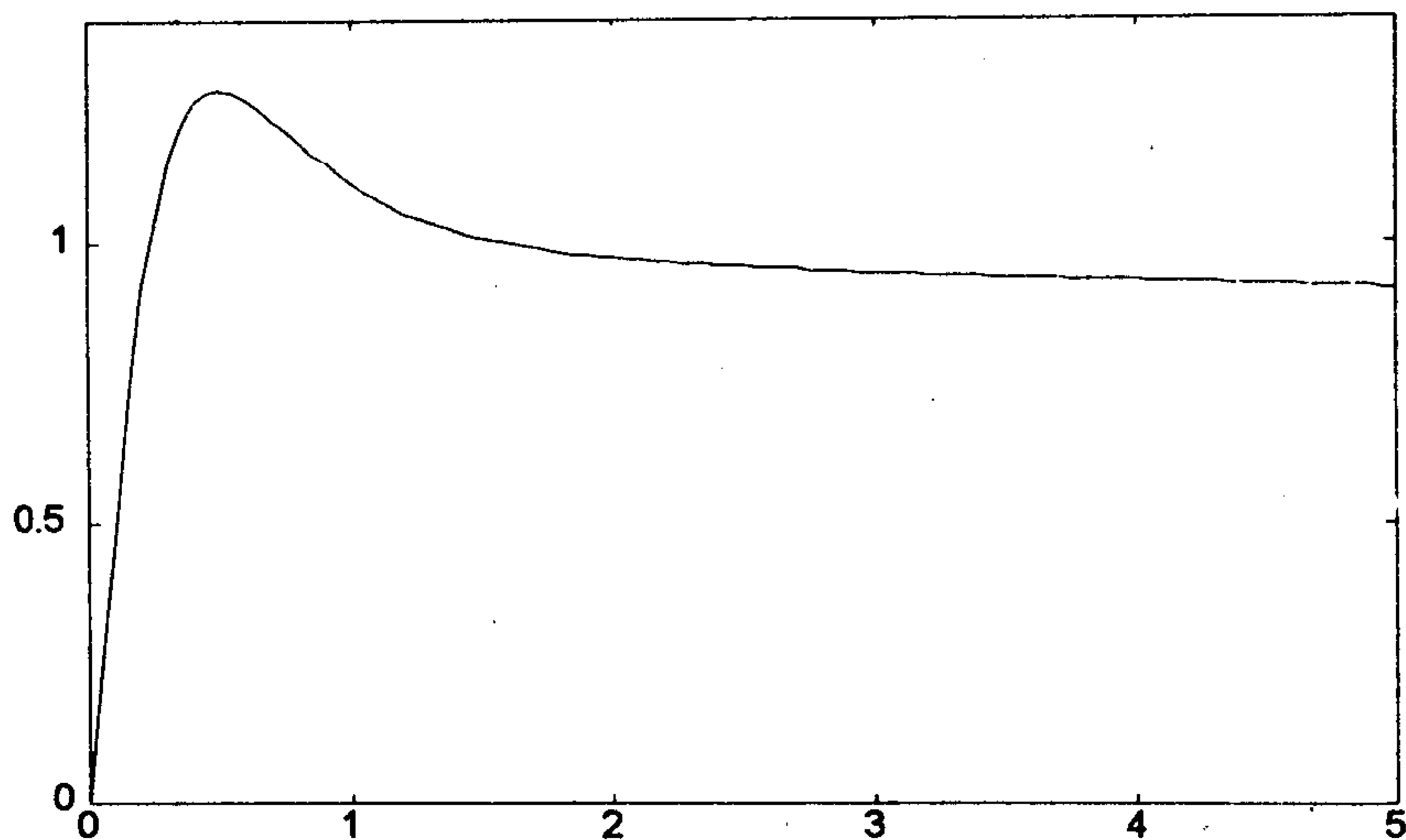


图7-17 LTR 设计的闭环系统阶跃响应曲线

率点处都将得出一个矩阵，所以就不再可以用如此简单的方法来描述系统的特性了。文献 [18] 中引入了主导增益 (principal gain) 的概念来描述这类问题，其思想是在每个频率点 ω 处传递函数矩阵中提取出其最大奇异值，记作 $\bar{\sigma}(\omega)$，这样由 $\bar{\sigma}(\omega)$ 随 ω 变化构成的曲线来描述系统的性能，这一曲线称为主导增益曲线，在单变量系统中无疑此曲线等效于 Bode 图形。

例 7.10 考虑双输入双输出系统的传递函数矩阵

$$G(s)=\begin{bmatrix}\dfrac{s+5}{s^2+2s+3} & \dfrac{1}{s+1}\\ \dfrac{1}{s+4} & \dfrac{1}{s+2}\end{bmatrix}$$

由下面的命令可以容易地绘制出系统的主导增益图形，如图 7-18 (a) 所示，定义灵敏度函数 $S(s)=(I+G)^{-1}$，则该函数的主导增益曲线如图 7-18(b) 所示。

```
>> num11=[1,5]; num12=[1]; num21=[1]; num22=[1];
>> den11=[1,2,3]; den12=[1,1]; den21=[1,4]; den22=[1,2]; I=eye(2);
>> w=logspace(-1,2); dly=zeros(2,2); p=2; q=2; newmv2fr
>> for i=1:length(w)
      V= mf(p*(i-1)+1:p*i,:); vv=svd(V); mag(i)=max(abs(vv));
      S=inv(V+I); vv=svd(S); magS(i)=max(abs(vv));
   end
>> subplot(121), semilogx(w,mag)
>> subplot(122), semilogx(w,magS)
```

除了由主导增益曲线来描述多变量系统之外，还可以引入单一的数值来对极端情形作出描述，比如对其范数作一描述，两个比较常用的指标定义如下

$$||G||_2=\sqrt{\frac{1}{2\pi}\int_{-\infty}^{\infty}\mathrm{tr}[G(j\omega)G^T(-j\omega)]\mathrm{d}\omega},\quad ||G||_\infty=\max_{\omega}\bar{\sigma}[G(j\omega)] \tag{7.5.1}$$

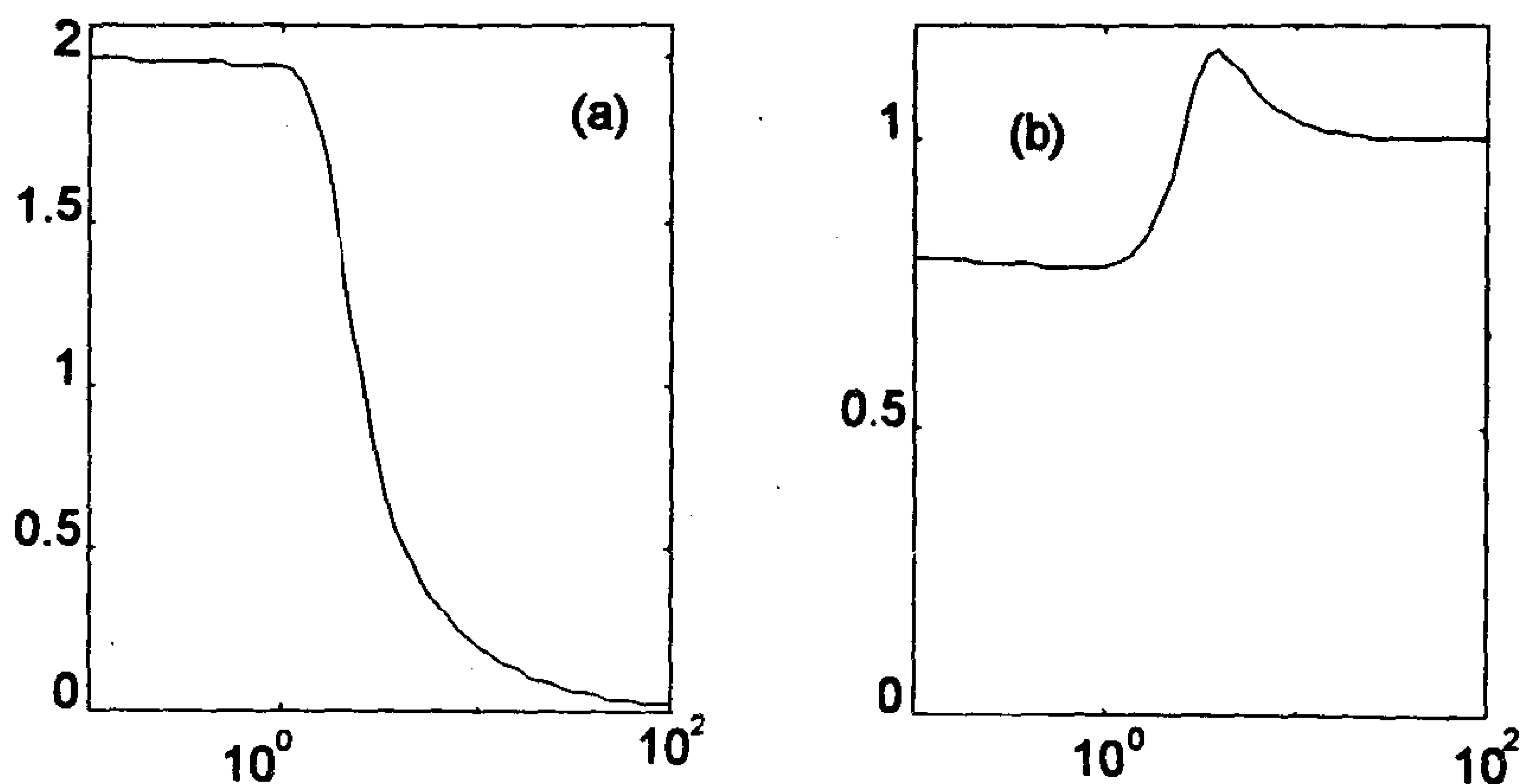


图7-18 主导增益曲线

若已知系统的状态空间模型 (A,B,C,D)，则可以由下式直接求出 $\|G\|_2$

$$\|G\|_2=\sqrt{CPC^T} \tag{7.5.2}$$

其中 P 为 Lyapunov 方程 $AP+PA^T=-BB^T$ 的解。求解 $\|G\|_\infty$ 就不再这样容易了，通常需要采用搜索的算法来求解。选择一个正数 γ，并构造一个 $2n\times 2n$ 矩阵 H

$$H=\begin{bmatrix} A & \gamma^{-2}BB^T \\ -C^TC & -A^T \end{bmatrix} \tag{7.5.3}$$

通过检验 H 矩阵是否在虚轴上存在特征值，依此来检验 $\|\gamma^{-1}G\|_\infty<1$ 是否满足，并相应地增大或减小 γ 的值 [7]。求取 $\|G(s)\|_\infty$ 通常可以由半分算法来进行搜索，最终得出一个收敛的 γ，这时可得系统的范数为 $\|G(s)\|_\infty=\gamma$。

MATLAB 的鲁棒控制工具箱提供了求解系统 $\|G\|_2$ 和 $\|G\|_\infty$ 的函数 normh2() 和 normhinf()，其调用格式是一致的

`normh2(A, B, C, D)` 和 `normhinf(A, B, C, D, tol)`

其中 A, B, C, D 为系统状态方程参数，tol 为搜索误差限，其默认值为 10^{-3}。

例 7.11 求出传递函数模型

$$G(s)=\frac{5s+100}{s^4+8s^3+32s^2+80s+100}$$

的 $\|G(s)\|_2$ 和 $\|G(s)\|_\infty$。直接调用鲁棒控制工具箱中的相应函数则可以立即得出系统的范数为

```
>> num=[5 100]; den=[1 8 32 80 100];
>> [a,b,c,d]=tf2ss(num,den);
>> normh2(a,b,c,d)
ans =  1.0765
>> normhinf(a,b,c,d)
ans =  1.0497
```

```
>> normhinf(a,b,c,d,1e-5)
ans =   1.0486
>> normhinf(a,b,c,d,1e-12)
ans =   1.0486
```

可见对 $\|G(s)\|_\infty$ 的求解确实还取决于误差限 tol 的选择，也可以用求解 Lyapunov 方程的方法得出系统的 $\|G(s)\|_2$ 参数

```
>> sqrt(c*lyap(a,b*b')*c')
ans =   1.0765
```

在控制系统的研究中经常要考虑如图 7-19 所示的结构，其中 $G(s)$ 为对象模型，

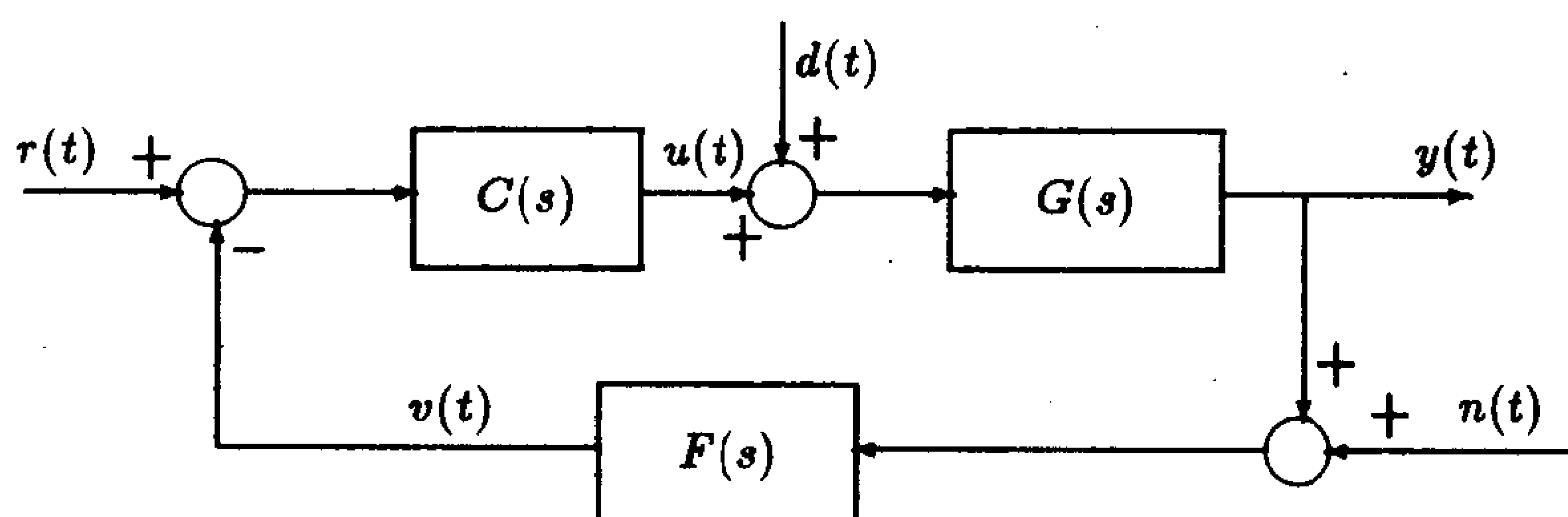


图7-19　典型反馈控制系统框图

$K(s)$ 为控制器模型，$F(s)$ 为反馈环节模型，信号 $r(t)$ 为输入信号，$d(t)$ 为扰动信号，而 $n(t)$ 为量测噪声信号，在实际控制中并不仅要求从输入 $r(t)$ 到输出 $y(t)$ 的闭环传递函数是稳定的，而且还要求从 $r(t),d(t),n(t)$ 到 $y(t),u(t)$ 和 $v(t)$ 所构成的 9 个闭环传递函数都是稳定的，这样才能保证系统中的信号均是有界的，否则可能会导致物理系统的内部结构的破坏，这种稳定性称作控制系统的内稳定 (internal stable)。判定系统内稳定的充要条件为特征多项式 $N_G N_K N_F + D_G D_K D_F = 0$ 的所有特征根均满足 $\Re e\ s < 0$，其中 N_G, D_G, 分别为受控对象的分子和分母多项式，而 N_K, D_K, N_F, D_F 分别为 $K(s)$ 和 $F(s)$ 的分子和分母多项式。

在控制理论文献中经常见到这样的记号：L_∞ 和 H_∞，在这里将叙述它们的意义和起源。L_∞ 表示全部满足 $\|G\|_\infty < \infty$ (即有界) 的物理可实现的正则函数集合，所谓物理可实现是指系统的传递函数分子阶次不大于分母阶次，亦简称为可实现的，因为这样的集合是一个 Lebesque 空间，故常常简记为 L_∞，而 H_∞ 则表示全部指数稳定的可实现正则函数集合，因为该集合是一个 Hardy 空间，故简记之为 H_∞。所谓指数稳定就是指正则传递函数矩阵 $G(s)$ 在闭右半 s 平面上没有极点，这和单变量系统稳定的条件是一致的，所以以后提起 H_∞ 就相当于指渐近稳定的可实现传递函数全体的集合。与此类似，前面介绍的 LQG 问题又常常被称为 H_2 问题。

描述反馈控制下不确定模型的框图如图 7-20(a) 所示，受控对象矩阵 $P(s)$ 可以按照 z 和 y 的形式作如下的分块

$$P(s) = \begin{bmatrix} P_{11}(s) & P_{12}(s) \\ P_{21}(s) & P_{22}(s) \end{bmatrix} \tag{7.5.4}$$

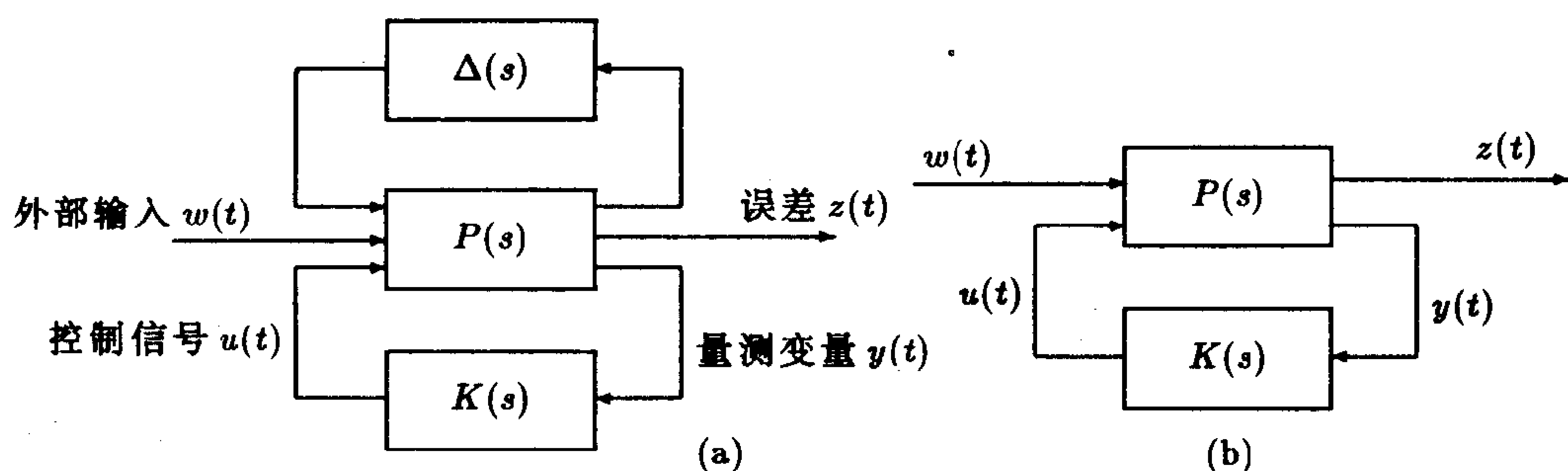


图 7-20 不确定模型的反馈控制标准描述

使得 $z = P_{11}w + P_{12}u$, $y = P_{21}w + P_{22}u$, 则引入反馈 $u = Ky$ 可消去 u 和 y, 最终得出

$$z = [P_{11} + P_{12}K(I - P_{22}K)^{-1}P_{21}]w = F_l(P, K)w \tag{7.5.5}$$

最优控制的目标就是使得 F_l 的范数极小化，亦即

$$\min_{K \in H_\infty} \|F_l(P, K)\|_\infty \tag{7.5.6}$$

这一问题称为 H_∞ 最小化问题，其含义是找出稳定的物理可实现的 K, 使得 F_l 的无穷范数趋于极小。

H_∞ 控制的研究起源于灵敏度最小化的例子 [25], 考虑如图 7-19 所示的系统框图，为简单起见不妨设 $F(s) = 1$, 控制的目的是求取控制器 $K(s)$ 使得系统的加权灵敏度为最小，即

$$\min \sup_{\omega} |W(j\omega)S(j\omega)| \tag{7.5.7}$$

其中 $S(s)$ 为灵敏度函数，可以写成 $S(s) = [I + G(s)K(s)]^{-1}$, 而 $W(s)$ 为加权函数，该函数在高频时的值较小，而在低频时其值趋于 1，其目的是对高频的信号作一定的约束。该函数的数学描述为

$$|W(j\omega)| \simeq 1, \ \ 0 \le \omega \le \omega_b, \ \ 且\ \ |W(j\omega)| \ll 1, \omega > \omega_b \tag{7.5.8}$$

其中 ω_b 为某一频率值。在下述的内容中将略去 (s) 字样。从上式可见，H_∞ 设计在很大程度上取决于加权函数的选择，所以有人认为 H_∞ 设计方法是选择加权矩阵的艺术 [15]。假设定义一个 Q 矩阵使得 $Q = K(I+GK)^{-1}$, 则可以得出 $S = I - GQ, (I+GK)^{-1} = I - QG$, 这样就可以得出下面的关系式

$$K = (I - QG)^{-1}Q \tag{7.5.9}$$

可见，若 Q 是稳定的，则图 7-19 所示的闭环系统将是稳定的，否则闭环系统将为不稳定的。可以证明，若 Q 为正则的，则控制器 K 亦为正则的 (即可实现的)，这样式 (7.5.9) 中定义的 K 就是全部可以使闭环系统稳定的可实现控制器的通式，这一公式就

是著名的 Youla 参数化 (Youla parametrisation) 公式，这时前面定义的最优准则就可以写成

$$\min_{Q\in H_\infty}\sup_{\omega}|W(I-GQ)|_{j\omega} \tag{7.5.10}$$

亦即使用了 Youla 参数化公式之后，原来的最优化问题就转换成了简单约束条件 (Q 为稳定可实现的) 下的最优化问题了，这会使得问题更容易进行求解。在下面一节中将进一步介绍各种情况下的 Youla 参数化公式和 H_∞ 问题的应用。

7.5.2 H_∞ 及镇定控制器参数化公式

前面给出的 H_∞ 公式 (7.5.6) 似乎很抽象，刚刚接触它的人一时往往找不到头绪，所以在这里将给出几个常见的例子来说明如何获得其 H_∞ 表示公式。

- **灵敏度最小化问题**: 前面涉及的灵敏度最小化问题可以表示成

$$F_l(P,K)=W(I+GK)^{-1}=W[I-GK(I+GK)^{-1}] \tag{7.5.11}$$

所以对照式 (7.5.6) 不难作出如下的矩阵分块表示

$$P_{11}=W,\ P_{12}=-WG,\ P_{21}=I,\ P_{22}=-G \tag{7.5.12}$$

这样就可以把灵敏度问题直接转换为 H_∞ 问题来求解了。

- **附加扰动的鲁棒控制问题**: 考虑图 7-21 (a) 所示的框图，注意，这里使用的全部是正反馈。系统的对象模型是由标称模型 $G(s)$ 和不确定扰动 $\Delta(s)$ 之和来表示的，且已

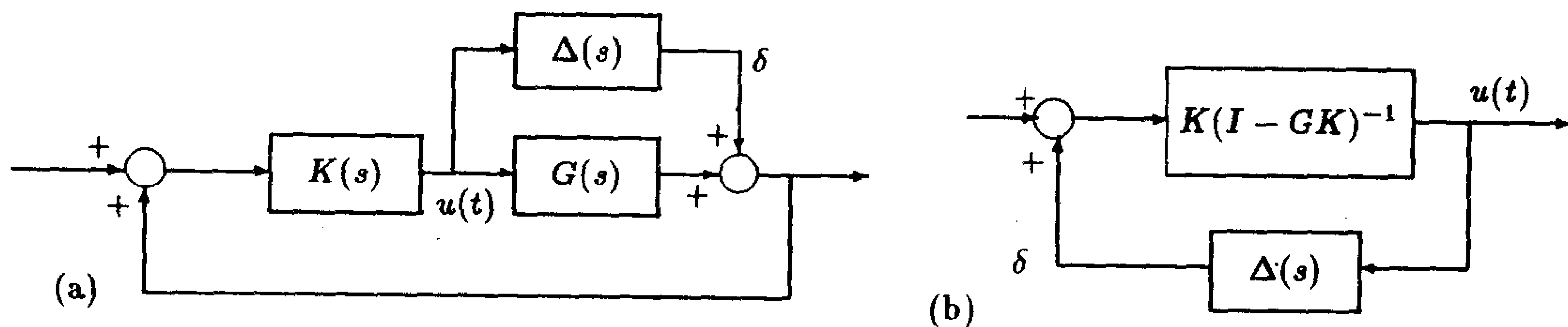


图 7-21　附加扰动鲁棒控制框图

知扰动 $\Delta(s)$ 增益的上界和频率有关，且满足

$$\bar{\sigma}[\Delta(j\omega)]<|r(j\omega)|,\ \text{对所有频率}\ \omega \tag{7.5.13}$$

这里 $r(s)$ 为标量传递函数。显而易见，这一描述是和第 6 章中介绍的 QFT 算法接近的，所以 H_∞ 可以解决和 QFT 同样的问题。此外还将假设对象模型 $G(s)+\Delta(s)$ 的右半平面极点个数在 $\Delta(s)$ 变化时保持为常值，对图中模型按照 δ 和 u 信号进行重新整理，则可以得出如图 7-21 (b) 所示的结构，这样可知若 $\|\Delta K(I-GK)^{-1}\|_\infty<1$，则系统回路将是稳定的。可以证明

$$\|\Delta K(I-GK)^{-1}\|_\infty\leq\|r^{-1}\Delta\|_\infty\|rK(I-GK)^{-1}\|_\infty<\|rK(I-GK)^{-1}\|_\infty \tag{7.5.14}$$

这样当 $\|rK(I-GK)^{-1}\| \le 1$ 则系统鲁棒稳定，为了使鲁棒稳定裕度尽可能大，则应该引入如下的最优化定义

$$\min \|rK(I-GK)^{-1}\|_\infty,\ K \text{ 为全部使系统稳定的控制器} \tag{7.5.15}$$

这时作下面的设置之后，则可以将此问题转换成标准的 H_∞ 问题

$$P_{11} = 0_{p\times m},\ P_{12} = rI_p,\ P_{21} = I_m,\ P_{22} = G \tag{7.5.16}$$

其中 p 和 m 分别为系统的输入和输出个数。

同时兼顾鲁棒性和品质 (performance) 的最优控制: 在保证系统鲁棒性的前提下若想改进系统的品质，则可以如下地定义最优控制的目标函数

$$\min_K F_l(P,K) = \min_K \begin{bmatrix} W_1 S \\ W_2(I-S) \end{bmatrix} \tag{7.5.17}$$

式中 S 为灵敏度函数，它和控制器 K 是有关的，若想建立起较有意义的指标，则加权矩阵 $W_1(s)$ 应该按照式 (7.5.8) 来选择，它可以保证系统高频时的灵敏度极小化，有关品质性能可以由加权矩阵 $W_2(s)$ 来设定，其选择方法为

$$|W_2(j\omega)| \ll 1, 0 \le \omega \le \omega_b,\ \text{且}\ |W_2(j\omega)| \simeq 1, \omega > \omega_b \tag{7.5.18}$$

亦即在低频部分对响应函数作出约束来改进品质特性，如果想把此问题转换为 H_∞ 的标准格式，则可以作如下 P 矩阵设置

$$P_{11} = \begin{bmatrix} W_1 \\ 0 \end{bmatrix},\ P_{12} = \begin{bmatrix} -W_1 G \\ W_2 G \end{bmatrix},\ P_{21} = I,\ P_{22} = -G \tag{7.5.19}$$

在介绍镇定控制器参数化公式之前，首先简要介绍矩阵多项式分式 (matrix-fraction description, 简称 MFD) 的描述与互质分解的概念。一般的多变量系统传递函数矩阵 $G(s)$ 都可以写成下面的形式

$$G(s) = N(s)D^{-1}(s) = \widetilde{D}^{-1}(s)\widetilde{N}(s) \tag{7.5.20}$$

其中 $N, D, \widetilde{N}$ 和 $\widetilde{D}$ 均为多项式矩阵，任何正则的传递函数矩阵 $G(s)$ 都可以表示成

$$G(s) = U(s)V^{-1}(s) = \widetilde{V}^{-1}(s)\widetilde{U}(s) \tag{7.5.21}$$

其中 $U, V, \widetilde{U}$ 和 $\widetilde{V}$ 都为稳定的传递函数矩阵，且 U 与 V 右互质 (right coprime)， $\widetilde{U}(s)$ 与 $\widetilde{V}(s)$ 左互质 (left coprime)，所谓右互质，则应该满足下面的 Bezout 定理：

矩阵 $U(s)$ 和 $V(s)$ 为右互质，当且仅当存在稳定的 $X(s)$ 和 $Y(s)$ 满足

$$U(s)X(s) + V(s)Y(s) = I \tag{7.5.22}$$

对左互质亦有相应的定理。注意， $X(s), Y(s), U(s), V(s)$ 等为稳定的传递函数矩阵，并不是多项式矩阵。若原系统模型是稳定的最小实现，则互质分解 Bezout 方程的解为 $U(s) = G(s)$, $V(s) = I$, $X(s) = 0, Y(s) = I$ 。若原系统为不稳定单变量模型，则可以按照下面的算法来进行互质分解：

- 利用映射 $s=(1-\lambda)/\lambda$ 将 $G(s)$ 变换成关于 λ 的传递函数 $\widetilde{G}(\lambda)$, 并将 $\widetilde{G}(\lambda)$ 写成互质多项式 $u(\lambda)$ 和 $v(\lambda)$ 之比 $u(\lambda)/v(\lambda)$,
- 利用代数方法求解满足 Bezout 方程的多项式 $x(\lambda)$ 和 $y(\lambda)$, 可参阅文献 [9],
- 采用映射 $\lambda=1/(s+1)$ 将 $u(\lambda),v(\lambda),x(\lambda),y(\lambda)$ 反变换回 $U(s),V(s),X(s),Y(s)$ 。

值得指出的是，由于前面算法中采用了 $\lambda=1/(s+1)$ 进行映射，所以这样的互质分解并不是唯一的，因为除了采用这样的映射外还可以采用 $\lambda=1/(s+a)$, $a\neq 1$ 或其它类型的映射来进行互质分解。下面将通过例子来说明不稳定传递函数的互质分解方法。

例 7.12　考虑不稳定模型 $G(s)=1/(s-1)(s-2)$，引入映射 $s=(1-\lambda)/\lambda$ 可以将原传递函数变换成 $\widetilde{G}(\lambda)=\lambda^2/(6\lambda^2-5\lambda+1)$, 故得出 $u(\lambda)=\lambda^2$, $v(\lambda)=6\lambda^2-5\lambda+1$ 。令多项式 $u(\lambda)$ 和 $v(\lambda)$ 分别可以分解成

$$u(\lambda)=q_1(\lambda)v(\lambda)+r_1(\lambda),\quad v(\lambda)=q_2(\lambda)r_1(\lambda)+r_2(\lambda)$$

则可以由多项式除法容易地得出商 $q_1(\lambda),q_2(\lambda)$ 和余式 $r_1(\lambda),r_2(\lambda)$

$$q_1(\lambda)=\frac{1}{6},\ r_1(\lambda)=\frac{5}{6}\lambda-\frac{1}{6},\ q_2(\lambda)=\frac{36}{5}\lambda-\frac{114}{25},\quad r_2(\lambda)=\frac{6}{25}=\text{常数}\ r_2$$

整理上面的式子可以容易地得出

$$v(\lambda)=[u(\lambda)-q_1(\lambda)v(\lambda)]q_2(\lambda)+r_2\quad\Rightarrow\quad -\frac{q_2(\lambda)}{r_2}u(\lambda)+\frac{1+q_1(\lambda)q_2(\lambda)}{r_2}v(\lambda)=1$$

即 $x(\lambda)=-q_2(\lambda)/r_2=-30\lambda+19$, $y(\lambda)=[1+q_1(\lambda)q_2(\lambda)]/r_2=5\lambda+1$ 。由映射 $\lambda=1/(s+1)$ 进行反变换则立即可以得出互质分解的子系统为

$$U(s)=\frac{1}{(s+1)^2},\ V(s)=\frac{(s-1)(s-2)}{(s+1)^2},\ X(s)=\frac{19s-11}{s+1},\ Y(s)=\frac{s+6}{s+1}$$

事实上，$U(s)$ 和 $V(s)$ 相当于同时除以一个稳定的多项式 $(s+1)^2$, 故最终可以分解成两个稳定的互质传递函数之商。

由前面的例子可见，对于高阶系统模型来说，用代数方法是很烦琐的，所以在实际应用中传递函数的互质分解还可以由状态空间的算法来处理，假设原系统模型的状态方程实现为 (A,B,C,D), 选择 F 和 H 矩阵分别使得 $A+BF$ 和 $A+HC$ 稳定，则可以由下面的式子得出互质分解后各个子系统的状态方程实现为

$$\begin{aligned}&U(s):(A+BF,B,C+DF,D),\qquad V(s):(A+BF,B,F,1)\\&X(s):(A+HC,H,F,0),\qquad Y(s):(A+HC,-B-HD,F,1)\end{aligned}\tag{7.5.23}$$

仍然考虑上例中的系统模型，采用极点配置的算法将 $A+BF$ 和 $A+HC$ 的极点均配置到 -1, -2, 则可以容易地得出可使子系统稳定的 F 和 H 向量，按照下面的方式得出系统的互质分解为

```
>> num=1; den=conv([1,-1],[1,-2]); [a,b,c,d]=tf2ss(num,den);
>> f=-acker(a,b,[-1;-1]')
f =    -5     1
>> h=-acker(a', c', [-1;-1]')
h =  -14     -5
```

```
>> ua=a+b*f; ub=b; uc=c+d*f; ud=d; [unum,uden]=ss2tf(ua,ub,uc,ud,1)
unum =  0     0     1
uden =  1     2     1
>> va=a+b*f; vb=b; vc=f; vd=1; [vnum,vden]=ss2tf(va,vb,vc,vd,1)
vnum =  1    -3     2
vden =  1     2     1
>> xa=a+h'*c; xb=h'; xc=f; xd=0; [xnum,xden]=ss2tf(xa,xb,xc,xd,1)
xnum =  0    65   -49
xden =  1     2     1
>> ya=a+h'*c; yb=-b-h'*d; yc=f; yd=1; [ynum,yden]=ss2tf(ya,yb,yc,yd,1)
ynum =  1     7    25
yden =  1     2     1
>> conv(unum,xnum)+conv(vnum,ynum)
ans =   1     4     6     4     1
>> conv(xden,uden)
ans =   1     4     6     4     1
```

可见采用状态空间的方法可以更方便地得出原系统的互质分解为

$$U(s)=\frac{1}{(s+1)^2},\ V(s)=\frac{s^2-3s+2}{(s+1)^2},\ X(s)=\frac{65s-49}{(s+1)^2},\ Y(s)=\frac{s^2+7s+25}{(s+1)^2}$$

这样得出的 $U(s)$ 和 $V(s)$ 结果与例 7.12 中的是一致的，因为这里仍假设分解后的极点均在 -1，然而即使在相同的假设下 $X(s)$ 和 $Y(s)$ 和前面得出的却是不同的，但它们仍满足 Bezout 方程。

我们将不加证明地给出下面的各个有关定理：

- 假设 G 和 K 可以由分式表示为 $G=NM^{-1}=\widetilde{M}^{-1}\widetilde{N}$ 和 $K=UV^{-1}=\widetilde{V}^{-1}\widetilde{U}$，则当且仅当下面两个矩阵

$$\begin{bmatrix} M & U \\ N & V \end{bmatrix}^{-1},\ \text{和}\ \begin{bmatrix} \widetilde{V} & -\widetilde{U} \\ -\widetilde{N} & \widetilde{M} \end{bmatrix}^{-1} \tag{7.5.24}$$

  均为稳定时，闭环反馈系统是稳定的。

- 如果系统是可稳定的 (stabilisable)，则可以按照下面方式选择 $M,N,U,V,\widetilde{M},\widetilde{N},\widetilde{U},\widetilde{V}$

$$\begin{bmatrix} \widetilde{V} & -\widetilde{U} \\ -\widetilde{N} & \widetilde{M} \end{bmatrix}\begin{bmatrix} M & U \\ N & V \end{bmatrix}=\begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \tag{7.5.25}$$

  且 $K=UV^{-1}=\widetilde{V}^{-1}\widetilde{U}$ 为镇定控制器 (stabilising controller)。

- 设 $K_0=U_0V_0^{-1}=\widetilde{V}_0^{-1}\widetilde{U}_0$ 为一个使得式 (7.5.25) 成立的控制器，则对任意的 $Q\in H_\infty$ 定义下列矩阵

$$U=U_0+MQ,\ V=V_0+NQ,\ \widetilde{U}=\widetilde{U}_0+Q\widetilde{M},\ \widetilde{V}=\widetilde{V}_0+Q\widetilde{N} \tag{7.5.26}$$

  则 $UV^{-1}=\widetilde{V}^{-1}\widetilde{U}$，且 $K=UV^{-1}=\widetilde{V}^{-1}\widetilde{U}$ 为使得 $G=NM^{-1}=\widetilde{M}^{-1}\widetilde{N}$ 镇定的控制器。此外全部镇定控制器均满足式 (7.5.26) 中的分式表示。

由前面给出的定理可知，如果已知其中一个镇定控制器 K_0, 则可以如下构造出一族镇定原系统的控制器 K 来

$$K = K_0 + \widetilde{V}_0^{-1}Q(I + V_0^{-1}NQ)^{-1}V_0^{-1} \tag{7.5.27}$$

这样就给出了设计 H_∞ 控制器的一种方法。下面将对稳定对象模型和不稳定对象模型分别介绍控制器的设计方法：

- **原对象 G 稳定**: 对此系统可以首先选择 $N = G, M = I$, 并令 $U_0 = \widetilde{U}_0 = 0$ 且 $V_0 = \widetilde{V}_0 = I$, 显然选择 $K_0 = 0$ 可以使得原系统稳定，这样就可以根据式 (7.5.27) 构造出控制器 $K = Q(I + GQ)^{-1}$, 从中可以反推出 $Q = K(I - GK)^{-1}$, 这就得出了灵敏度极小化例子中的 Youla 参数化公式。这时就可以按照图 7-22 (a) 来表示闭环控制系统的框图了，可以看出在此系统结构下，控制器 K 是由 Q 与 $-G$ 的正反馈连接而构

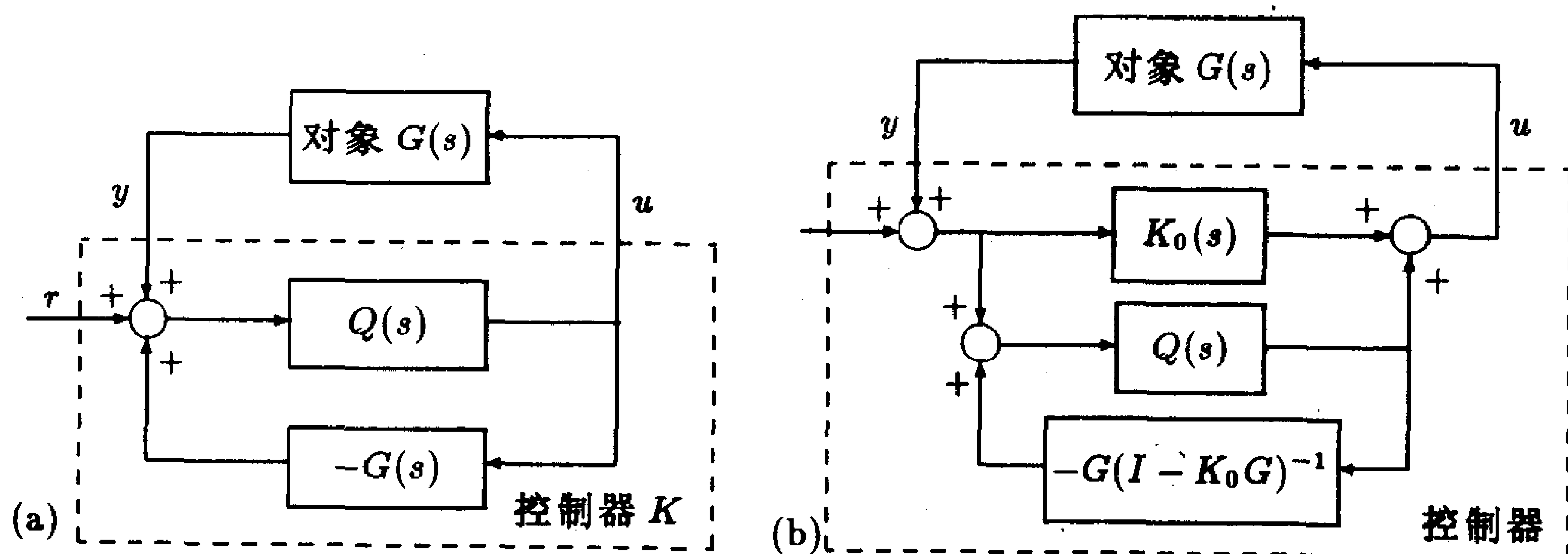


图 7-22 镇定控制器结构的框图表示

成的，其意义可以理解为用 $-G$ 来抵销原系统的特性，并加入任意的稳定系统 Q 来修正系统的动态特性。

- **原对象 G 不稳定**: 若原系统模型不稳定，则不难首先想到按照图 7-22(a) 的思路来构造控制器，但用不稳定的 $-G$ 来抵销 G 显然是不合适的。其实对不稳定的 G 若想设计镇定的控制器可以通过以下两个步骤，其中第一个步骤首先设计一个稳定的控制器 K_0 来镇定原系统，然后在第二个步骤中采用上面的方法来对稳定系统作出设计，这样就可以得出如图 7-22 (b) 所示的系统框图。可以看出从 u 到 y 的等效传递函数为 $H = G(I - K_0G)^{-1}$, 而由互质分解可见

$$H = NM^{-1}(I - \widetilde{V}_0^{-1}\widetilde{U}_0NM^{-1})^{-1} = N(M - \widetilde{V}_0^{-1}\widetilde{U}_0N)^{-1} = N\widetilde{V}_0 \tag{7.5.28}$$

若 K_0 为稳定的，则可以引入 $R = \widetilde{V}_0^{-1}QV_0^{-1}$, 可以证明 $R \in H_\infty$ 当且仅当 $Q \in H_\infty$, 这时 Q 和 K 可以分别写成

$$Q = \widetilde{V}_0RV_0, \quad K = K_0 + R(I + N\widetilde{V}_0R)^{-1} \tag{7.5.29}$$

在一些特定的情况下不稳定对象模型 G 只允许由不稳定的控制器 K_0 来镇定，这样图 7-22 (b) 中的控制结构就失效了，而应该采用图 7-23 (a) 中所示的分式方式来实现，

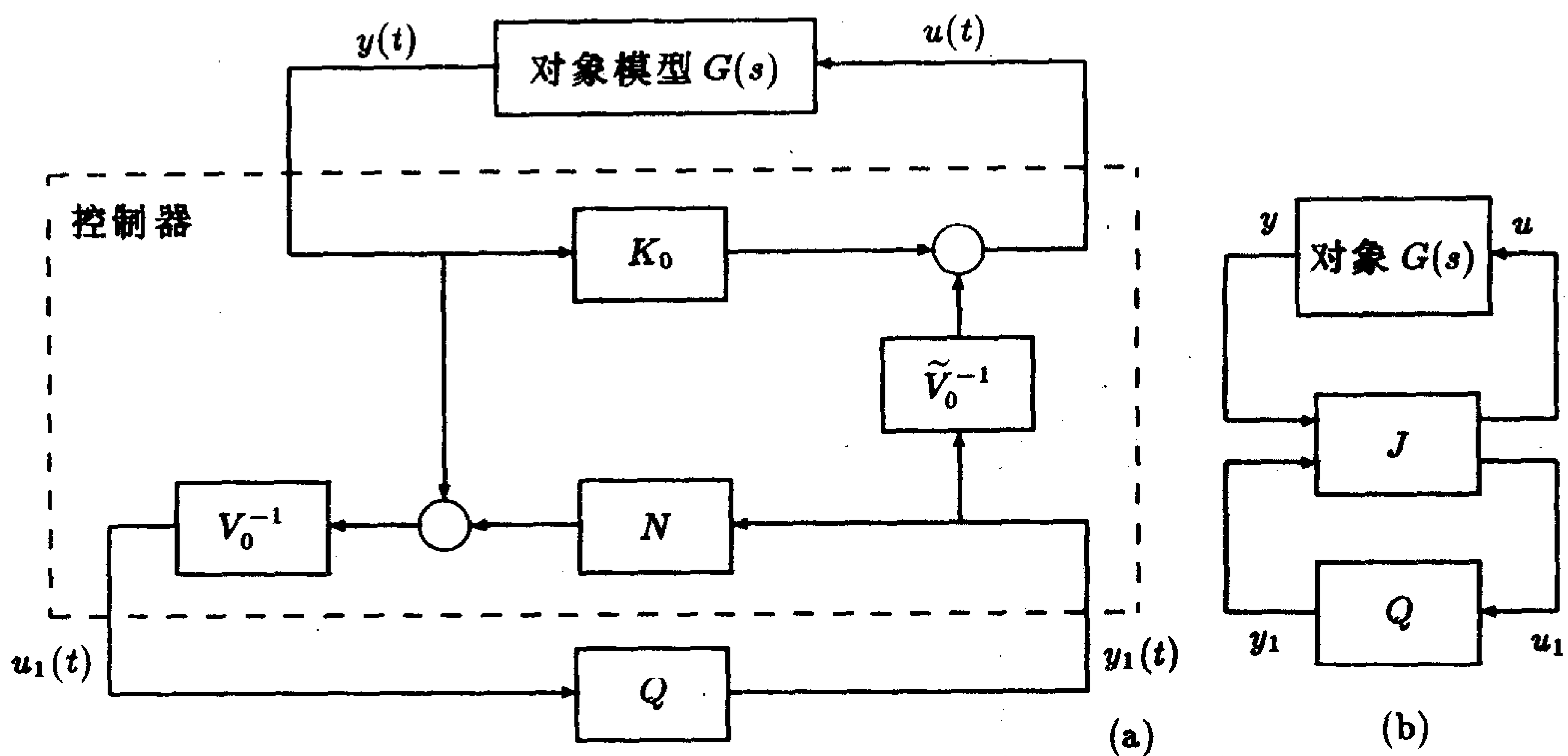


图7-23 $Q \in H_\infty$ 的一般镇定控制器

并用 J 来代替其中的虚线框内部分，则可以将该系统简单地表示成图 7-23 (b) 中的形式，这时框图和图 7-20 (a) 中的类似，故可以得出 F_l 和 J 表示

$$K = F_l(J,Q), \quad J = \begin{bmatrix} K_0 & \tilde{V}_0^{-1} \\ V_0^{-1} & -V_0^{-1}N \end{bmatrix} \tag{7.5.30}$$

从上面的叙述中若使 Q 在 H_∞ 中变化，就可以容易地得出全部可能的镇定控制器 K 来。下面将讨论控制器的状态方程实现问题。由于系统的状态一般是不能直接测取的，所以在实际控制中往往需要通过观测器来获得系统的状态变量，这种问题又常常被称为基于观测器的 (observer-based) 控制问题。假设系统 $G = NM^{-1}$ 的状态方程实现为 (A,B,C,D)，并假设 F 是可以使得 $A+BF$ 渐近稳定的任意矩阵 (例如 F 可以是状态反馈问题的解)，则 G 可以由下面的状态方程来实现

$$\dot{x} = (A+BF)x + Bv, \quad u = Fx + v, \quad y = (C+DF)x + Dv \tag{7.5.31}$$

这样就可以得出稳定的 $N(s)$ 和 $M(s)$ 传递函数矩阵

$$M(s) = F(sI - A - BF)^{-1}B + I, \quad N(s) = (C+DF)(sI - A - BF)^{-1}B + D \tag{7.5.32}$$

选择可以使得 $A+HC$ 稳定的任意矩阵 H (例如求解观测器问题)，并考虑观测器方程

$$\dot{\zeta} = (A+HC)\zeta - Hy + (B+HD)u, \quad \eta = C\zeta + Du - y \tag{7.5.33}$$

则可以得出稳定的 $\widetilde{M}(s)$ 和 $\tilde{N}(s)$ 传递函数矩阵

$$\widetilde{M}(s) = -C(sI - A - HC)^{-1}H + I \tag{7.5.34}$$

$$\tilde{N}(s) = C(sI - A - HC)^{-1}(B + HD) + D \tag{7.5.35}$$

类似地还可以得出如下实现

$$V_0^{-1}:(A+BF+HC+HDF,H,C+DF,I),\quad \tilde{V}_0^{-1}:(A+BF+HC+HDF,B+HD,F,I) \tag{7.5.36}$$

$$V_0^{-1}N:\left(\begin{bmatrix} A+BF+HC+HDF & H(C+DF) \\ 0 & A+BF \end{bmatrix},\begin{bmatrix} HD \\ B \end{bmatrix},[C+DF,C+DF],D\right) \tag{7.5.37}$$

由于上面的实现是不可观测的，所以可以简化成

$$V_0^{-1}N:(A+BF+HC+HDF,B+HD,C+DF,D) \tag{7.5.38}$$

这时可以得出 J 的实现

$$J:\left(A+BF+HC+HDF,[-H,B+HD],\begin{bmatrix} F \\ -(C+DF) \end{bmatrix},\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}\right) \tag{7.5.39}$$

假设 J 的输入向量为 $[y^T,y_1^T]$ 且输出向量为 $[u^T,u_1^T]$，则根据上式可以写出下面的控制器状态方程

$$\dot{\zeta}=(A+BF+HC+HDF)\zeta-Hy+(B+HD)y_1,\quad u=F\zeta+y_1,\quad u_1=-(C+DF)\zeta+y-Dy_1 \tag{7.5.40}$$

此外已知 $y_1=Qu_1$，所以该控制器状态方程可以如图 7-24 来表示。

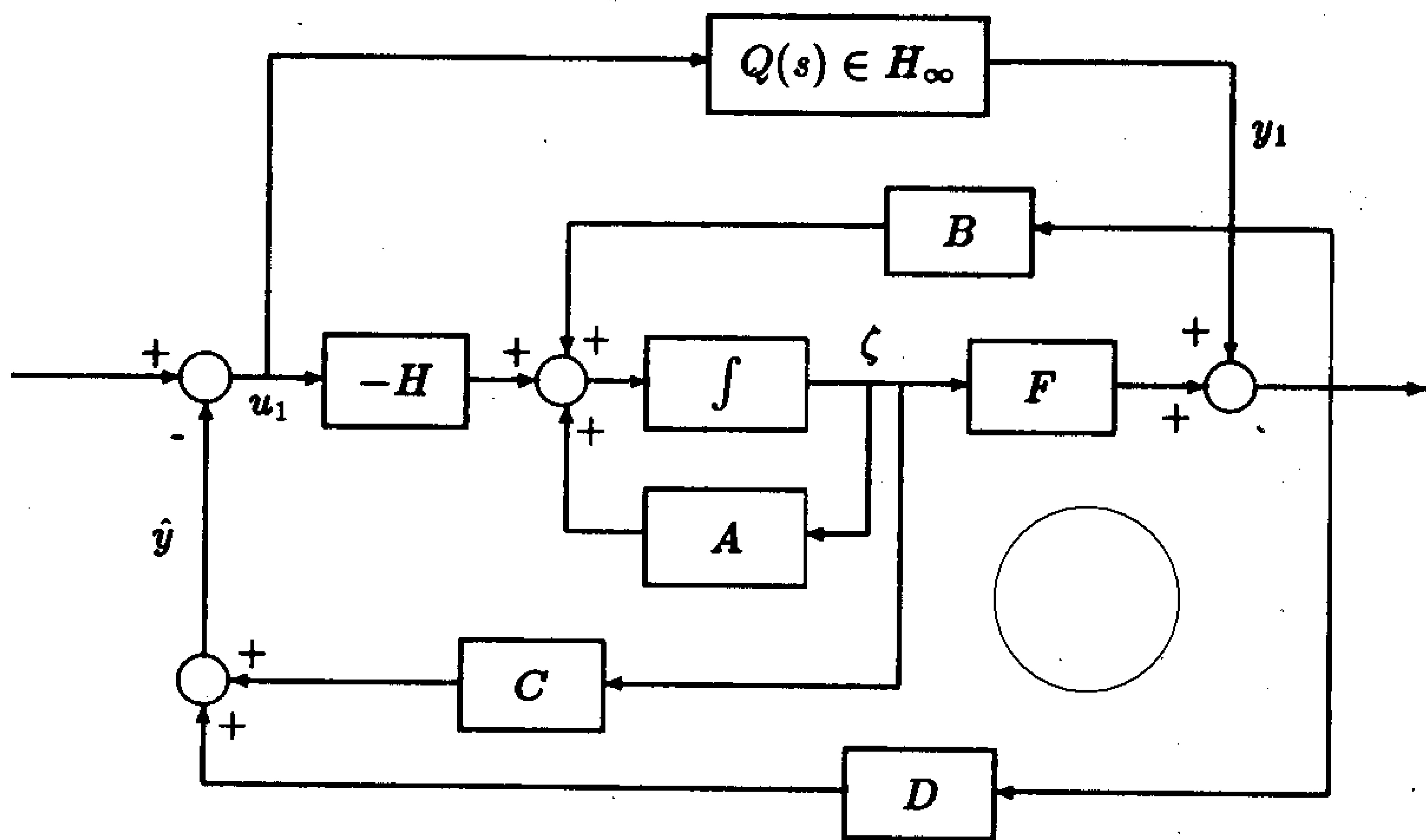


图7-24 镇定控制器的状态空间表示

假设 Q 的实现为

$$\dot{x}_q=A_qx_q+B_qu_1,\quad y_1=C_qx_q+D_qu_1 \tag{7.5.41}$$

且对象模型为严格正则 (strictly proper)，亦即 $D=0$，则消去前面式中的 u_1 和 y_1 可以得出

$$\begin{aligned} \dot{\zeta}&=(A+BF+HC-BD_qC)\zeta+BC_qx_q+(BD_q-H)y \\ \dot{x}_q&=A_qx_q-B_qC\zeta+B_qy,\quad u=(F-D_qC)\zeta+C_qx_q+D_qy \end{aligned} \tag{7.5.42}$$

记 $x_e = \begin{bmatrix} \zeta \\ x_q \end{bmatrix}, A_e = \begin{bmatrix} A & 0 \\ 0 & A_q \end{bmatrix}, B_e = \begin{bmatrix} B \\ 0 \end{bmatrix}, C_e = [C, 0], F_e = [F, C_q], H_e = \begin{bmatrix} H \\ -B_q \end{bmatrix}$, 则式 (7.5.42) 可以改写成

$$\dot{x}_e = (A_e + B_eF_e + H_eC_e - B_eD_eC_e)x_e + (B_eD_q - H_e)y,\ \ u = (F_e - D_qC_e)x_e + D_qy \quad (7.5.43)$$

其框图表示如图 7-25 所示，该方程就是控制器 K 的实现公式。注意这时 K 为严格正则当且仅当 Q 为严格正则。

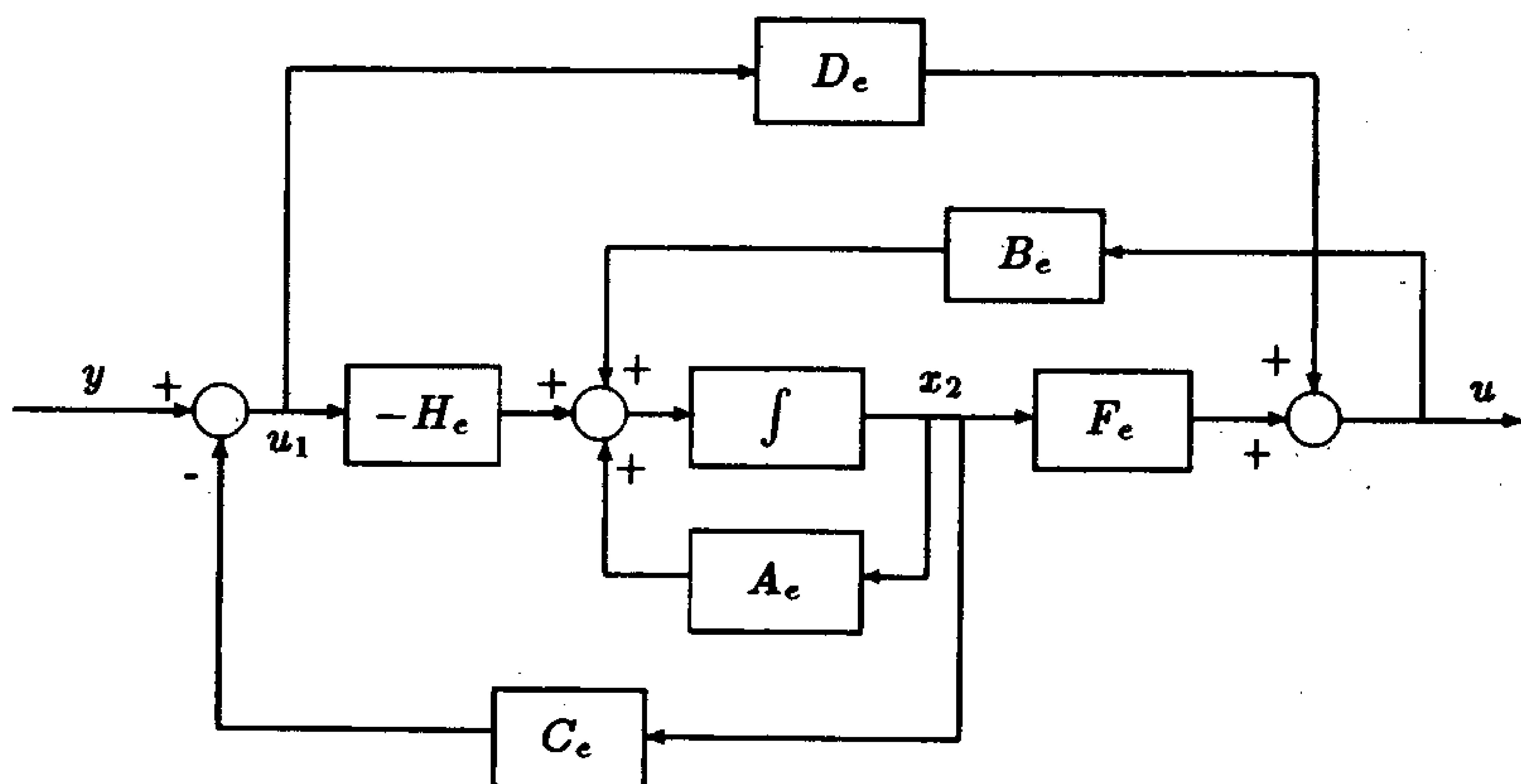


图 7-25　严格正则对象模型的全部镇定控制器

下面将讨论闭环系统的参数化公式，仿照图 7-20 (a), 令 $P_{12} = G$, 则可以构造出如图 7-26 (a) 所示的框图，再将其中的 $J(s)$ 和 $P(s)$ 的反馈结构进行合并，就可以将该系统

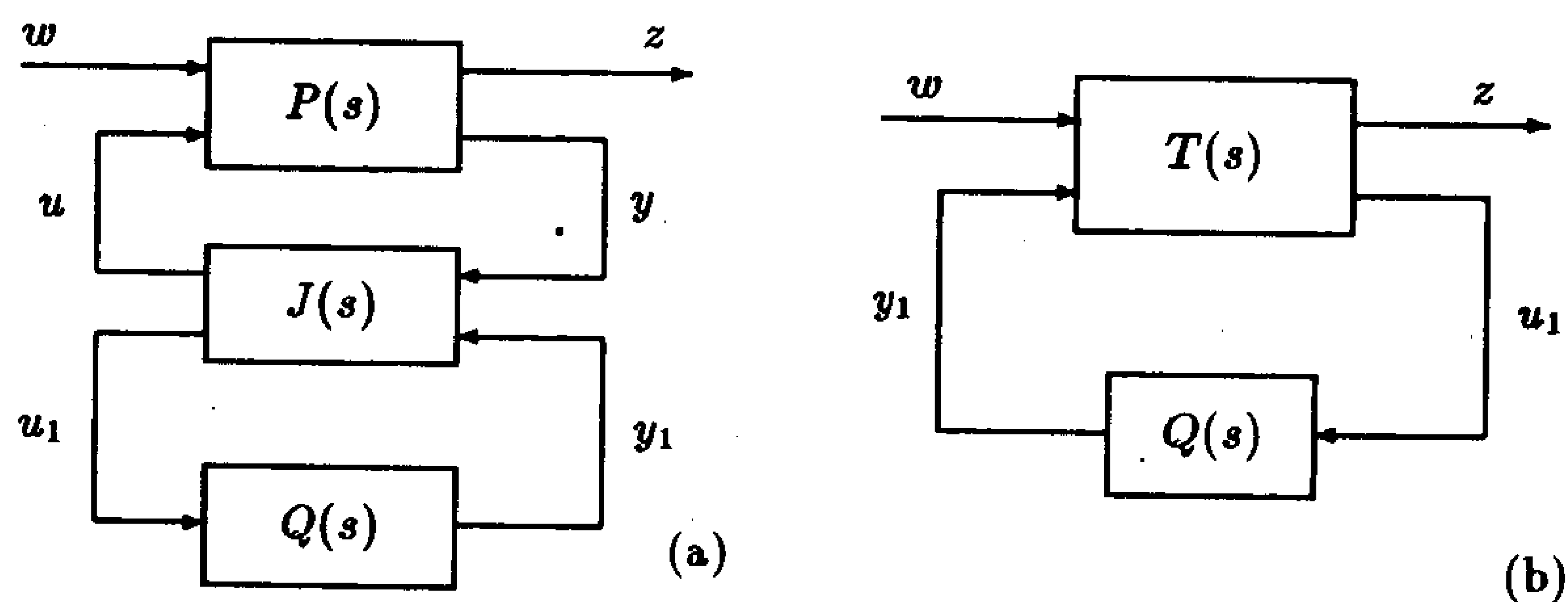


图 7-26　控制系统的闭环结构

简化成图 7-26 (b) 所示的框图结构，这里 $T(s)$ 即为 P 和 J 的反馈连接，这时可以得出

$$z = F_l(P, K)w = F_l(T, Q)w = [T_{11} + T_{12}Q(I - T_{22}Q)^{-1}T_{21}]w \quad (7.5.44)$$

可以证明 $T_{22} = 0$, 这样上式可以简化成

$$z = [T_{11} + T_{12}QT_{21}]w \quad (7.5.45)$$

且闭环系统 T 满足

$$T_{11} = P_{11} - P_{12}U_0\widetilde{M}P_{21},\ T_{12} = -P_{12}M,\ T_{21} = \widetilde{M}P_{21},\ T_{22} = 0 \quad (7.5.46)$$

其中 P 的状态方程实现为

$$P:\ \left(A,[B_1,B_2],\begin{bmatrix}C_1\\C_2\end{bmatrix},\begin{bmatrix}D_{11}&D_{12}\\D_{21}&D_{22}\end{bmatrix}\right) \tag{7.5.47}$$

亦即 $P_{ij}(s)=C_i(sI-A)^{-1}B_j+D_{ij},\ i,j=1,2$ 。这时可以证明若 F 为对象模型的镇定状态反馈矩阵 (即 $A+B_2F$ 稳定)，而 H 为对象的镇定观测器增益矩阵 (即 $A+HC_2$ 稳定)，则 $T_{ij}(s)$ 的状态方程实现为

$$\begin{aligned}&T_{11}:\ \left(\begin{bmatrix}A+B_2F&-B_2F\\0&A+HC_2\end{bmatrix},\begin{bmatrix}B_1\\B_1+HD_{21}\end{bmatrix},[C_1+D_{12}F,-D_{12}F],D_{11}\right)\\&T_{12}:\ (A+B_2F,B_2,C_1+D_{12}F,D_{12}),\quad T_{21}:\ (A+HC_2,B_1+HD_{21},C_2,D_{21})\end{aligned} \tag{7.5.48}$$

7.5.3 控制系统的鲁棒性能设计算法

考虑前面涉及到的最小灵敏度指标，参见式 (7.5.7)。我们的目的是设计一个控制器，使得在保证系统内稳定的前提下，达到

$$\|W(s)S(s)\|_\infty<1 \tag{7.5.49}$$

根据控制器参数化公式，若对象传递函数可以互质分解为 $G(s)=U(s)/V(s)$ 且由 Bezout 方程求出两个传递函数 $X(s)$ 和 $Y(s)$，则使得反馈系统达到内稳定的所有控制器 $K(s)$ 的集合为

$$\left\{\frac{X(s)+V(s)Q(s)}{Y(s)-U(s)Q(s)}:\ \ Q(s)\in H_\infty\right\} \tag{7.5.50}$$

更简单地，若原系统 $G(s)$ 稳定，则将互质分解的结果 $U(s)=G(s),V(s)=1,X(s)=0,Y(s)=1$ 代入控制器集合，则可以简化控制器的集合为

$$\left\{\frac{Q(s)}{1-G(s)Q(s)}:\ \ Q(s)\in H_\infty\right\} \tag{7.5.51}$$

所以若能得出一个合适的 $Q(s)$ 则可以设计出一个满意的控制器 $K(s)$，可见设计 $Q(s)$ 就成为控制器设计的关键。这里将分下面 3 种情况来考虑镇定控制器 $K(s)$ 的设计算法：

- $G(s)$ 与 $G^{-1}(s)$ 均为稳定的，则

  1) 选择一个足够小的 τ，使得 $\|W(s)[1-J(s)]\|_\infty<1$ 成立，其中 $J(s)=1/(\tau s+1)^k$ 且 k 为对象模型的相对阶次 (即分母阶次与分子阶次之差)， $J(s)$ 是为使得 $Q(s)$ 传递函数正则而引入的，
  2) 令 $Q(s)=G^{-1}(s)J(s)$，
  3) 可以得出镇定控制器为 $K(s)=Q(s)/[1-P(s)Q(s)]$ 。

- $G(s)$ 不稳定而 $G^{-1}(s)$ 稳定，则

1) 首先可以作互质分解，使得

$$G(s)=\frac{U(s)}{V(s)},\quad U(s)X(s)+V(s)Y(s)=1$$

且 $U(s)\in H_\infty, V(s)\in H_\infty, X(s)\in H_\infty, Y(s)\in H_\infty$

2) 选择一个足够小的 τ, 使得 $\|W(s)V(s)Y(s)[1-J(s)]\|_\infty<1$ 成立，

3) 令 $Q(s)=Y(s)U^{-1}(s)J(s)$,

4) 可以得出镇定控制器为 $K(s)=[X(s)+V(s)Q(s)]/[Y(s)-U(s)Q(s)]$ 。

- 若 $G^{-1}(s)$ 不稳定，则

1) 首先对系统进行互质分解

$$G(s)=\frac{U(s)}{V(s)},\quad U(s)X(s)+V(s)Y(s)=1$$

且 $U(s)\in H_\infty, V(s)\in H_\infty, X(s)\in H_\infty, Y(s)\in H_\infty$,

2) 求出一个稳定的 $Q_{\rm im}(s)$ 满足 $\|W(s)V(s)[Y(s)-U(s)Q_{\rm im}(s)]\|_\infty<1$,

3) 选择一个足够小的 τ, 使得 $\|W(s)V(s)[Y(s)-U(s)Q_{\rm im}(s)J(s)]\|_\infty<1$ 成立，

4) 令 $Q(s)=Q_{\rm im}(s)J(s)$,

5) 可以得出镇定控制器为 $K(s)=[X(s)+V(s)Q(s)]/[Y(s)-U(s)Q(s)]$ 。

例 7.13 假设不稳定对象环节 $G(s)=1/(s-2)^2$, 选择一个加权函数 $W(s)=100/(s+1)$, 使得系统的带宽为 1 rad/s, 且具有较好的跟踪性能 (因为 $\omega=1$ 时的跟踪误差为 1%), 这样依前面所述的第二种情况可以得出 $k=2$, 且互质分解得出

$$U(s)=\frac{1}{(s+1)^2},\ V(s)=\frac{(s-2)^2}{(s+1)^2},\ X(s)=\frac{27(s-1)}{s+1},\ Y(s)=\frac{s+7}{s+1}$$

对 τ 的不同取值，可以容易地获得 $\|W(s)[1-J(s)]\|_\infty$ 的值为

```
>> tau0=logspace(-1,-4,10); num0=100*conv(conv([1 -2],[1 -2]),[1,7]);
>> den0=conv(conv(conv([1,1],[1,1]),[1,1]),[1,1]); norms=[];
>> for tau=tau0
      n0=conv(num0, [tau^2, 2*tau, 0]);
      d0=conv(conv([tau,1],[tau,1]),den0);
      [a,b,c,d]=tf2ss(n0,d0);
      norms=[norms, normhinf(a,b,c,d,1e-5)];
   end
>> [tau0; norms]
ans = Columns 1 through 7
     0.1000     0.0464     0.0215     0.0100     0.0046     0.0022     0.0010
   199.0079    92.6408    43.0274    19.9744     9.2716     4.3036     1.9975
      Columns 8 through 10
     0.0005     0.0002     0.0001
     0.9272     0.4304     0.1997
```

可见，τ 取值为 0.0005, 0.0002 以及 0.0001 时均有 $\|W(s)[1-J(s)]\|_\infty < 1$, 分别取这 3 个 τ 值来构造 $Q(s)$ 模型，$Q(s)=(s+1)(s+7)/(\tau s+1)^2$,

$$K(s)=\frac{27(s-1)(\tau s+1)^2+(s-2)^2(s+7)}{\tau s(s+7)(\tau s+2)}$$

代入 3 个 τ 值，则可以得出 3 个镇定控制器，并可绘制出它们构成的闭环系统阶跃响应如图 7-27 所示。

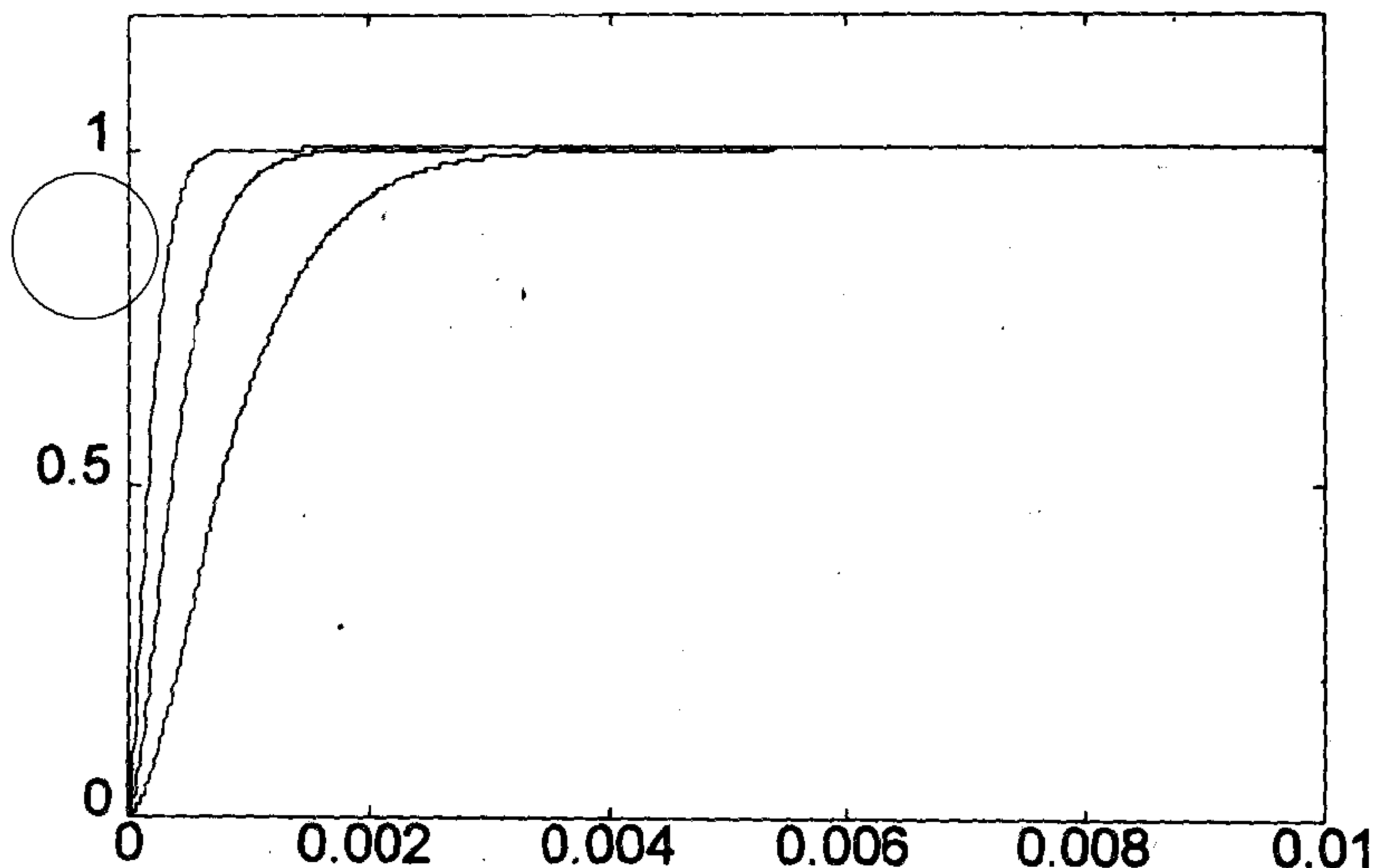


图7-27　不同 τ 值下的闭环阶跃响应比较

```
>> format long
>> NK1=27*conv([1,-1],conv([tau0(8),1],[tau0(8),1]))+conv([1,7],[1,-4,4])
NK1 =  1.00000581697366   3.02505876272785   2.97493542029849   1.00000000000000
>> DK1=tau0(8)*[conv([1,7],[tau0(8),2]), 0]
DK1 =  0.00000021544347   0.00092982587101   0.00649822436706               0
>> NK2=27*conv([1,-1],conv([tau0(9),1],[tau0(9),1]))+conv([1,7],[1,-4,4]);
>> DK2=tau0(9)*[conv([1,7],[tau0(9),2]), 0];
>> NK3=27*conv([1,-1],conv([tau0(10),1],[tau0(10),1]))+conv([1,7],[1,-4,4]);
>> DK3=tau0(10)*[conv([1,7],[tau0(10),2]), 0];
>> t=0:0.00001:0.01; DD1=conv(DK1,[1,-4,4])+[0,0,NK1]; y1=step(NK1,DD1,t);
>> DD2=conv(DK2,[1,-4,4])+[0,0,NK2]; y2=step(NK2,DD2,t);
>> DD3=conv(DK3,[1,-4,4])+[0,0,NK3]; y3=step(NK3,DD3,t);
>> plot(t,y1,t,y2,t,y3)
```

可见，闭环系统的阶跃响应跟踪性能非常好且 τ 的值越小越好，事实上，若 τ 的值特别小，则控制器可以写成

$$K(s)=\lim_{\tau\to 0}\frac{27(s-1)(\tau s+1)^2+(s-2)^2(s+7)}{\tau s(s+7)(\tau s+2)}\to\frac{K_c(s+1)^3}{s(s+7)(\tau s+2)}$$

且 $K_c\to\infty$, $K(s)$ 趋于非正则，故在实际应用中 τ 不能太小。

我们可以进一步探索 $\tau=10^{-4}$ 时镇定控制器的控制效果，考虑对象模型 $\widetilde{G}(s)=1/(s-a)^2$, 其中 a 可以从 0.01 变化到 10, 这时可以得出如图 7-28 所示的控制效果。可见若 a 参数允许发生这样的

变化，则从中取 $a=2$ 作为标称对象，就可以设计一个控制器 $K(s)$，它可以在对象环节有较大幅度变化时仍然可以稳定，且可以获得理想的时域特性。

```
>> t=0:0.00001:0.01; y=[];
>> for a=logspace(-2,1,30);
      DDx=conv(DK1,[1,-2*a a*a])+[0,0,NK1]; y=[y step(NK1,DDx,t)];
   end
>> plot(t,y)
```

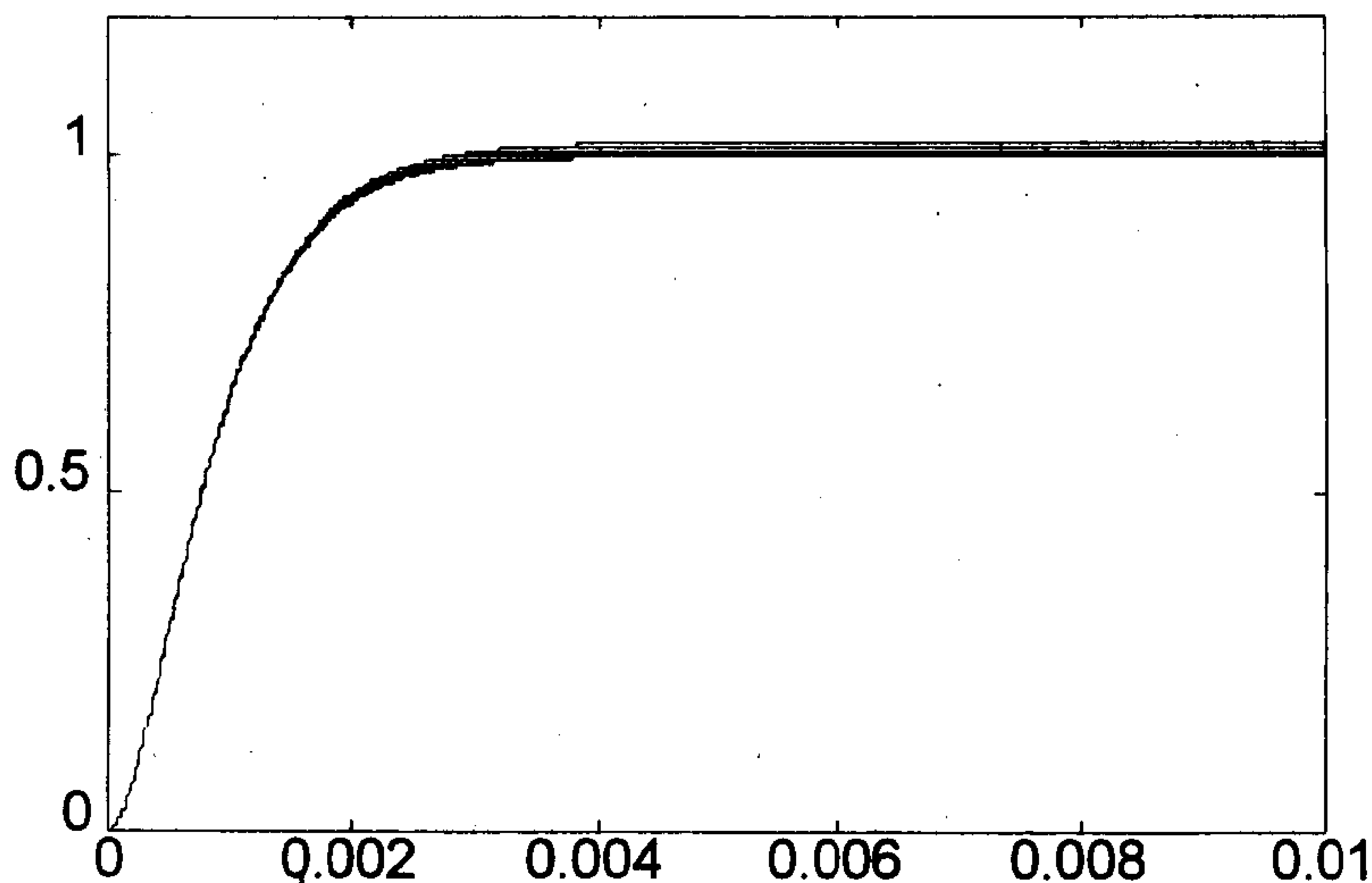


图7-28　不同 a 值下的闭环阶跃响应比较

7.5.4　H_∞ 问题的间接解法

所谓间接解法就是首先将 H_∞ 最优控制问题转换成模型匹配问题，然后通过求解降阶模型的方法来获得 H_∞ 最优控制问题的解。对传递函数矩阵 $X(s)$，若记 $X^*(s)=X^T(-s)$，亦即 $X^*(s)$ 为 $X(s)$ 的共轭转置矩阵，并假设 T_{12} 和 T_{21} 为方阵且满足 $T_{12}T_{12}^*=I, T_{21}^*T_{21}=I$，则可以得出下面的等效关系

$$\|T_{11}+T_{12}QT_{21}\|_\infty=\|T_{12}(T_{12}^*T_{11}T_{21}^*+Q)T_{21}\|_\infty=\|T_{21}T_{11}^*T_{12}+Q^*\|_\infty \tag{7.5.52}$$

因为 $Q\in H_\infty$，$T_{ij}\in H_\infty$，且 Q^* 只含有不稳定极点，而 $R=T_{21}T_{11}^*T_{21}$ 既可以含有稳定极点，又可以含有不稳定极点，所以 H_∞ 最优化准则满足

$$\min_{Q\in H_\infty}\|T_{11}+T_{12}QT_{21}\|_\infty=\min_{Q\in H_\infty}\|R+Q^*\|_\infty \tag{7.5.53}$$

亦即将原问题转化为求取原始传递函数矩阵 R 不稳定近似 $-Q^*$ 的模型匹配问题，而此问题又常常称为 Hankel 近似问题 (Hankel approximation problem) 或 Nehari 扩展问题 (Nehari extension problem)，这里只讨论 R 为稳定的情形。文献 [13] 给出了找出全部 L_∞ 降阶模型的算法：

- 对原始模型 R 进行均衡实现并得出 (A,B,C,D) 和对角矩阵 Σ, 满足

$$A\Sigma+\Sigma A^T+BB^T=0,\ A^T\Sigma+\Sigma A+C^TC=0 \tag{7.5.54}$$

记 $\Sigma=\mathrm{diag}(\sigma_1 I,\Sigma_1)$, 这时 $\Sigma_1=\mathrm{diag}(\sigma_2,\cdots,\sigma_n)$, 且 $\sigma_1>\sigma_2\geq\cdots\geq\sigma_n>0$。

- 根据 Σ 的分块方式对 A,B,C 进行分块，使得

$$A=\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix},\ B=\begin{bmatrix}B_1\\ B_2\end{bmatrix},\ C=[C_1,C_2] \tag{7.5.55}$$

令 $\Gamma=\Sigma_1^2-\sigma_1^2 I$, 并选择正交矩阵 U 使得 $UU^T=I$, 且 $B_1=-C_1^TU$, 可以证明这样的矩阵总是存在，因为 $B_1B_1^T=C_1^TC_1$ 。

- 得出下面的稳定降阶模型 $Q_{\rm opt}$ 为 $(\widetilde{A},\widetilde{B},\widetilde{C},\widetilde{D})$:

$$\begin{aligned}&\widetilde{A}=\Gamma^{-1}(\sigma_1^2A_{22}^T+\Sigma_1A_{22}\Sigma_1-\sigma_1C_2^TUB_2^T),\ \widetilde{B}=\Gamma^{-1}(\Sigma_1B_2+\sigma_1C_2^TU)\\ &\widetilde{C}=-C_2\Sigma_1-\sigma_1UB_2^T,\ \widetilde{D}=-D+\sigma_1U\end{aligned} \tag{7.5.56}$$

这时最优降阶模型 $Q_{\rm opt}^*$ 的状态方程实现为 $(-\widetilde{A}^T,-\widetilde{C}^T,\widetilde{B}^T,\widetilde{D}^T)$。

可以证明，由上面方法得出的 $Q_{\rm opt}$ 满足 [13]

$$Q_{\rm opt}\in H_\infty,\ \|R+Q_{\rm opt}^*\|_\infty=\sigma_1,\ (R+Q_{\rm opt}^*)(R+Q_{\rm opt}^*)^*=\sigma_1^2I \tag{7.5.57}$$

可见上述的过程需要求取系统的均衡实现，而文献 [22] 提出了一种更简单的算法来求取 Q^*, 首先求出系统的可控和可观测 Gram 矩阵 P 和 S, 满足

$$AP+PA^T+BB^T=0,\quad A^TS+SA+C^TC=0 \tag{7.5.58}$$

注意，和式 (7.5.54) 中的 (A,B,C,D) 不同，这里的矩阵为原矩阵，而不是均衡实现后的矩阵。由第 4 章可知，这样得出的 P 和 S 都是对称矩阵。任意指定一个 $m\geq\sigma_1$, 则令 $\Gamma_m=SP-m^2I$, 可以得出下面各个矩阵

$$\widetilde{A}=\Gamma_m^{-1}(m^2A^T-SAP),\ \widetilde{B}=\Gamma_m^{-1}SB,\ \widetilde{C}=-CP,\ \widetilde{D}=-D \tag{7.5.59}$$

可见这样就从算法上大大地简化了文献 [13] 中的算法。

例 7.14　在这里将演示稳定模型的 Hankel 近似问题，假设系统的传递函数为

$$G(s)=\frac{24}{s^4+10s^3+35s^2+50s+24}$$

则可以由该传递函数构造出系统的状态方程模型，并调用鲁棒控制工具箱中的 `ohklmr()` 函数来求出系统的 Hankel 近似模型。

```
>>  num=24; den=[1,10,35,50,24]; [A,B,C,D]=tf2ss(num,den);
>>  [Ah,Bh,Ch,Dh]=ohklmr(A,B,C,D,1,2)
2    states removed !!
Ah = -0.7927    0.9620
     -0.3248   -0.8073
Bh =-11.3719
     65.7376
Ch =  0.0179   -0.0012
Dh =  0
```

该系统相应的不稳定模型可以如下求出。

```
>>  As=-Ah', Bs=-Ch', Cs=-Bh'
As =  0.7927    0.3248
     -0.9620    0.8073
Bs = -0.0179
      0.0012
Cs = 11.3719  -65.7376
```

7.5.5 H_∞ 问题的直接解法

可以看出，在前面方法中需要首先将标准的 H_∞ 问题转换为 Hankel 最优近似的问题，然后才可以对之进行求解，最后得出原来问题的解。目前解决 H_∞ 最优化问题多使用状态空间方法，因为采用这样的方法是不需要将原始问题进行转换的，所以这类方法又称为直接方法。文献 [14] 中提出了对选定常数 γ 获得一族满足

$$\|F_l(P,K)\|_\infty < \gamma \tag{7.5.60}$$

的镇定控制器 K 的算法，假设原系统模型的状态方程实现可以由式 (7.5.47) 给出，且 P_{11}, P_{12}, P_{21} 和 P_{22} 的维数分别为 $p_1\times m_1$, $p_1\times m_2$, $p_2\times m_1$ 和 $p_2\times m_2$, 并进一步假设对象模型 (A,B_2,C_2,D_{22}) 为可稳定的及可检测的 (detectable), 使得镇定控制器存在。

为保证控制器的稳定性假设 $\mathrm{rank}\{D_{12}\}=m_2$ 且 $\mathrm{rank}\{D_{21}\}=p_2$ 。为进一步简化原始问题，则可以引入非奇异变换，使得 $\tilde{u}=S_u u$, $\tilde{y}=S_y y$, 其中 S_u; S_y 为非奇异矩阵，并引入变换 $\tilde{w}=T_w w$, $\tilde{z}=T_z z$, 其中 $T_w T_w^H = T_z T_z^H = I$, 这时将得出

$$\begin{bmatrix}\tilde{z}\\ \tilde{y}\end{bmatrix}=\begin{bmatrix}T_zP_{11}T_w^H & T_zP_{12}S_u^{-1}\\ S_yP_{21}T_w^H & S_yP_{22}S_u^{-1}\end{bmatrix}\begin{bmatrix}\tilde{w}\\ \tilde{u}\end{bmatrix}=\begin{bmatrix}\widetilde{P}_{11} & \widetilde{P}_{12}\\ \widetilde{P}_{21} & \widetilde{P}_{22}\end{bmatrix}\begin{bmatrix}\tilde{w}\\ \tilde{u}\end{bmatrix} \tag{7.5.61}$$

这样对控制器 $\widetilde{K}=S_uKS_y^{-1}$ 可以写出 $\tilde{w}=F_l(\widetilde{P},\widetilde{K})\tilde{z}=[T_wF_l(P,K)T_y^H]\tilde{z}$, 所以可见 $\|F_l(\widetilde{P},\widetilde{K})\|_\infty=\|F_l(P,K)\|_\infty$ 。若将原系统的 P 矩阵换为 $\widetilde{P}$ 矩阵，则可以设计出一个控制器 $\widetilde{K}$, 这样就可以由 $K=S_u^{-1}\widetilde{K}S_y$ 来得出原系统的控制器。为简化起见可以选择 T_z 和 S_u 使得 $\widetilde{D}_{12}=T_zD_{12}S_u^{-1}=[0,I]^T$, 并选择 T_w 和 S_y 使得 $\widetilde{D}_{21}=S_yD_{21}T_w^H=[0,I]$, 这

一工作可以由奇异值分解技术来完成。在这里我们不再使用 $\tilde{D}$ 类记号，而假设系统已经过上述的变换，故有 $D_{12}=[0,I]^T$, $D_{21}=[0,I]$ 。假设 D_{11} 可以分解成

$$\begin{bmatrix} \{D_{11}\}_{11} & \{D_{11}\}_{12} \\ \{D_{11}\}_{21} & \{D_{11}\}_{22} \end{bmatrix}$$

其中 $\{D_{11}\}_2$ 为 $m_2 \times p_2$ 矩阵。为推导方便暂时假设系统为严格正则的，这样有 $D_{22}=0$ 。引入下面的记号 $D_{1*}=[D_{11},D_{12}], D_{*1}=[D_{11}^T,D_{21}^T]^T$, 并定义

$$R=D_{1*}^TD_{1*}-\begin{bmatrix}\gamma^2 I_{m_1} & 0\\ 0 & 0\end{bmatrix},\quad \tilde{R}=D_{*1}D_{*1}^T-\begin{bmatrix}\gamma^2 I_{p_1} & 0\\ 0 & 0\end{bmatrix} \tag{7.5.62}$$

设 $\Gamma_x=A-BR^{-1}D_{1*}^TC_1$, $\Gamma_y=A-B_1D_{*1}^T\tilde{R}^{-1}C$, 定义 X_∞ 与 Y_∞ 为下面的 Riccati 方程的镇定解

$$\begin{aligned} &X_\infty\Gamma_x+\Gamma_x^TX_\infty-X_\infty BR^{-1}B^TX_\infty+C_1^T(I-D_{1*}R^{-1}D_{1*}^T)C_1=0\\ &Y_\infty\Gamma_y^T+\Gamma_yY_\infty-Y_\infty C^T\tilde{R}^{-1}CY_\infty+B_1(I-D_{*1}^T\tilde{R}^{-1}D_{*1})B_1^T=0 \end{aligned} \tag{7.5.63}$$

所谓镇定解，即得出的 X_∞ 和 Y_∞ 按照式 (7.5.64) 构造出的 F 和 H 矩阵可以使得 $A+BF$ 和 $A+HC$ 的所有特征值的极点都位于左半开平面

$$F=-R^{-1}(D_{1*}^TC_1+B^TX_\infty),\quad H=-(B_1D_{*1}^T+Y_\infty C^T)\tilde{R}^{-1} \tag{7.5.64}$$

注意，这里构造的 Riccati 方程和前面遇到的有所不同，因为前面一直假定 Riccati 方程二次项和常数项的矩阵为半正定矩阵，而这里的矩阵却不满足这些条件，所以不能直接沿用前面的方法去求解。假设可以将式 (7.5.64) 分块表示为 $F=[F_{11}^T,F_{12}^T,F_2^T]^T$, $H=[H_{11},H_{12},H_2]$, 这里 F_{11}, F_{12} 和 F_2 分别有 m_1-p_2, p_2 和 m_2 行，且 H_{11}, H_{12} 和 H_2 分别有 p_1-m_2, m_2 和 p_2 列。这时可以证明

- 存在使得 $\|F_l(P,K)\|_\infty<\gamma$ 成立的镇定控制器，当且仅当 $\gamma>\max(\bar{\sigma}[\{D_{11}\}_{11},\{D_{11}\}_{12},])$ 且存在使 $\rho(X_\infty,Y_\infty)<\gamma^2$ 成立的 $X_\infty\geq 0$ 与 $Y_\infty\geq 0$ 。
- 若镇定控制器存在，则满足 $\|F_l(P,K)\|_\infty<\gamma$ 的全部镇定控制器 K 为 $K=F_l(K_a,\Phi)$, 其中 $\Phi\in H_\infty$ 使得 $\|\Phi\|_\infty<\gamma$, 而 K_a 的状态方程实现为

$$K_a:\ \left(\hat{A},\begin{bmatrix}\hat{B}_1 & \hat{B}_2\end{bmatrix},\begin{bmatrix}\hat{C}_1\\ \hat{C}_2\end{bmatrix},\begin{bmatrix}\hat{D}_{11} & \hat{D}_{12}\\ \hat{D}_{21} & 0\end{bmatrix}\right) \tag{7.5.65}$$

且 $\hat{D}_{11}=-\{D_{11}\}_{21}\{D_{11}\}_{11}^T\left(\gamma^2I-\{D_{11}\}_{11}\{D_{11}\}_{11}^T\right)^{-1}\{D_{11}\}_{12}-\{D_{11}\}_{22}$, 且 $\hat{D}_{12}$ 和 $\hat{D}_{21}$ 分别为满足

$$\begin{aligned} &\hat{D}_{12}\hat{D}_{12}^T=I-\{D_{11}\}_{21}\left(\gamma^2I-\{D_{11}\}_{11}^T\{D_{11}\}_{11}\right)^{-1}\{D_{11}\}_{21}^T\\ &\hat{D}_{21}^T\hat{D}_{21}=I-\{D_{11}\}_{12}^T\left(\gamma^2I-\{D_{11}\}_{11}\{D_{11}\}_{11}^T\right)^{-1}\{D_{11}\}_{12} \end{aligned} \tag{7.5.66}$$

的任意 $m_2 \times m_2$ 和 $p_2 \times p_2$ 方阵，式 (7.5.65) 中的其它矩阵可以由下式得出

$$\begin{aligned}
&\widehat{B}_2 = (B_2 + H_{12})\widehat{D}_{12}, \quad \widehat{C}_2 = -\widehat{D}_{21}(C_2 + F_{12})(I - \gamma^2 Y_\infty X_\infty)^{-1} \\
&\widehat{B}_1 = -H_2 + \widehat{B}_2\widehat{D}_{12}^{-1}\widehat{D}_{11}, \quad \widehat{C}_1 = F_2(I - \gamma^2 Y_\infty X_\infty)^{-1} + \widehat{D}_{11}\widehat{D}_{21}^{-1}\widehat{C}_2 \\
&\widehat{A} = A + HC + \widehat{B}_2\widehat{D}_{12}^{-1}\widehat{C}_1
\end{aligned} \tag{7.5.67}$$

若令 $\Phi \equiv 0$, 则可以得出这类控制器中的一族简单控制器 K, 其实现为

$$K : (\widehat{A}, \widehat{B}_1, \widehat{C}_1, \widehat{D}_{11}) \tag{7.5.68}$$

它和 P 的维数是一致的。这类控制器又称为中间熵 (central-entropy) 或最大熵 (maximum-entropy) 控制器，当然对 Φ 进行其它的假设也可以得出其它形式的控制器，例如可以得出最小熵 (minimum-entropy) 控制器等，在这里就不再加以介绍了。

回顾前面的假设可知，前面一直假定 $D_{22} = 0$, 若此条件不满足，则可以通过图 7-29 所示的框图对之进行变换，首先构造一个新的 $\widehat{P}$ 矩阵

$$\widehat{P} = P - \begin{bmatrix} 0 & 0 \\ 0 & D_{22} \end{bmatrix} \tag{7.5.69}$$

这样 $\widehat{P}$ 为严格正则的系统，依照前面的算法设计出新的控制器 $\widehat{K}$, 这时原系统 P 的控制器可以容易地由

$$K = \widehat{K}(I + D_{22}\widehat{K})^{-1} \tag{7.5.70}$$

构造出来。

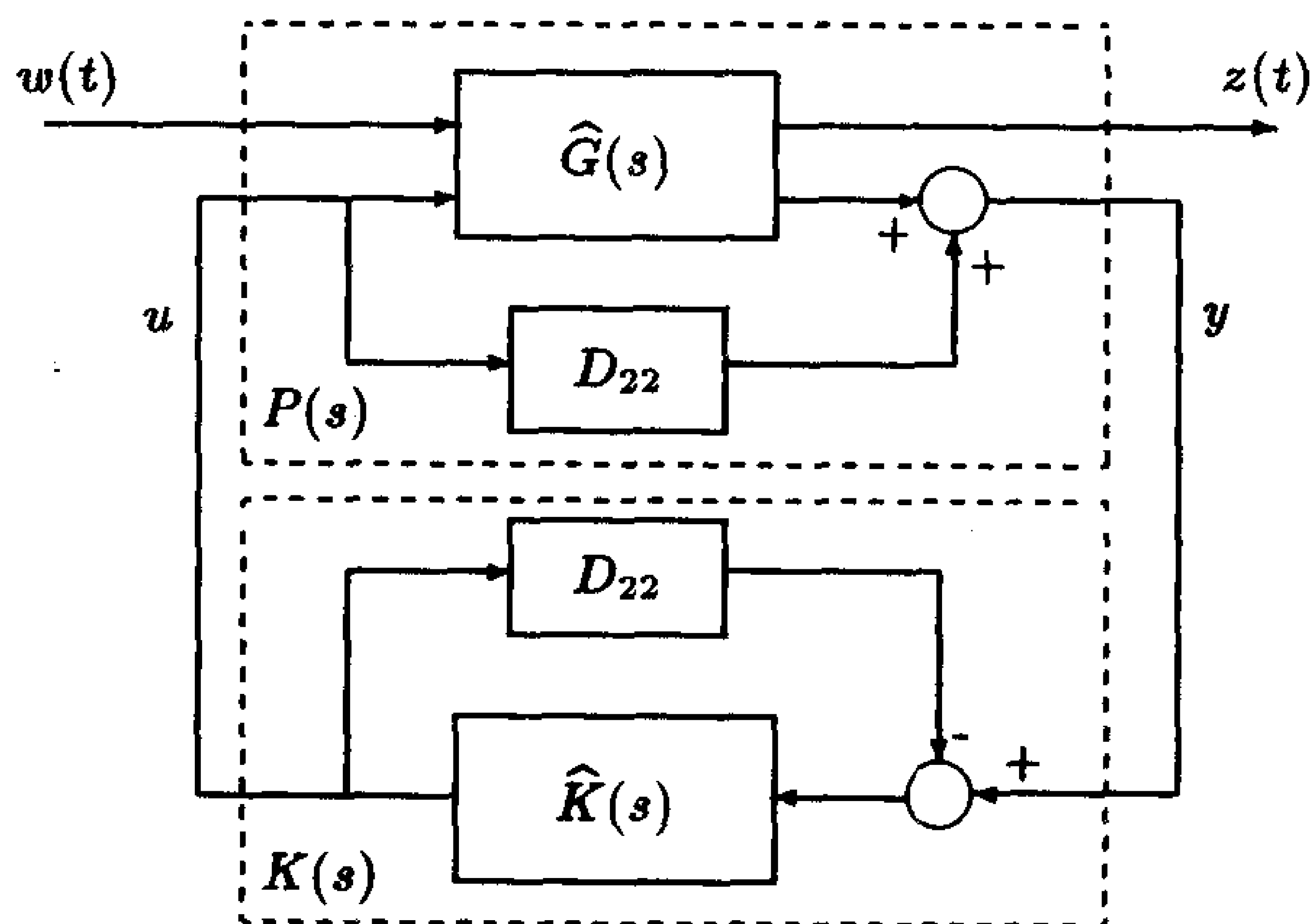


图 7-29　非严格有理函数的变换框图

为获得使得 $\|F_l(P,K)\|_\infty$ 最小化的控制器，则首先应该选定一个 γ, 获得 (X_∞, Y_∞) 对，逐步减小 γ 的值，直至 $\rho(X_\infty, Y_\infty) = \gamma_0^2$ 或式 (7.5.63) 中至少一个 Riccati 方程有半正定解，这样就可以得出在极限 γ_0 处的控制器实现，这时就已经获得系统的最优 H_∞ 控制器。

7.5.6 H_∞ 控制器的实现与降阶

由前面的方法得出的 H_∞ 最优控制器阶次往往是相当高的，实现起来一般存在困难，所以经常需要使用较低阶的控制器来近似地实现它们。文献 [1] 探讨了 H_∞ 低阶控制器设计的几种途径，如图 7-30 所示，其中对高阶对象模型来说共有 3 种途径可循，第一种途径是直接给它设计出一个低阶的控制器来，但对 H_∞ 控制器来说，这种方法一般是不可行的。第二种途径就是首先对原高阶系统进行某种降阶近似，然后根据降阶模型

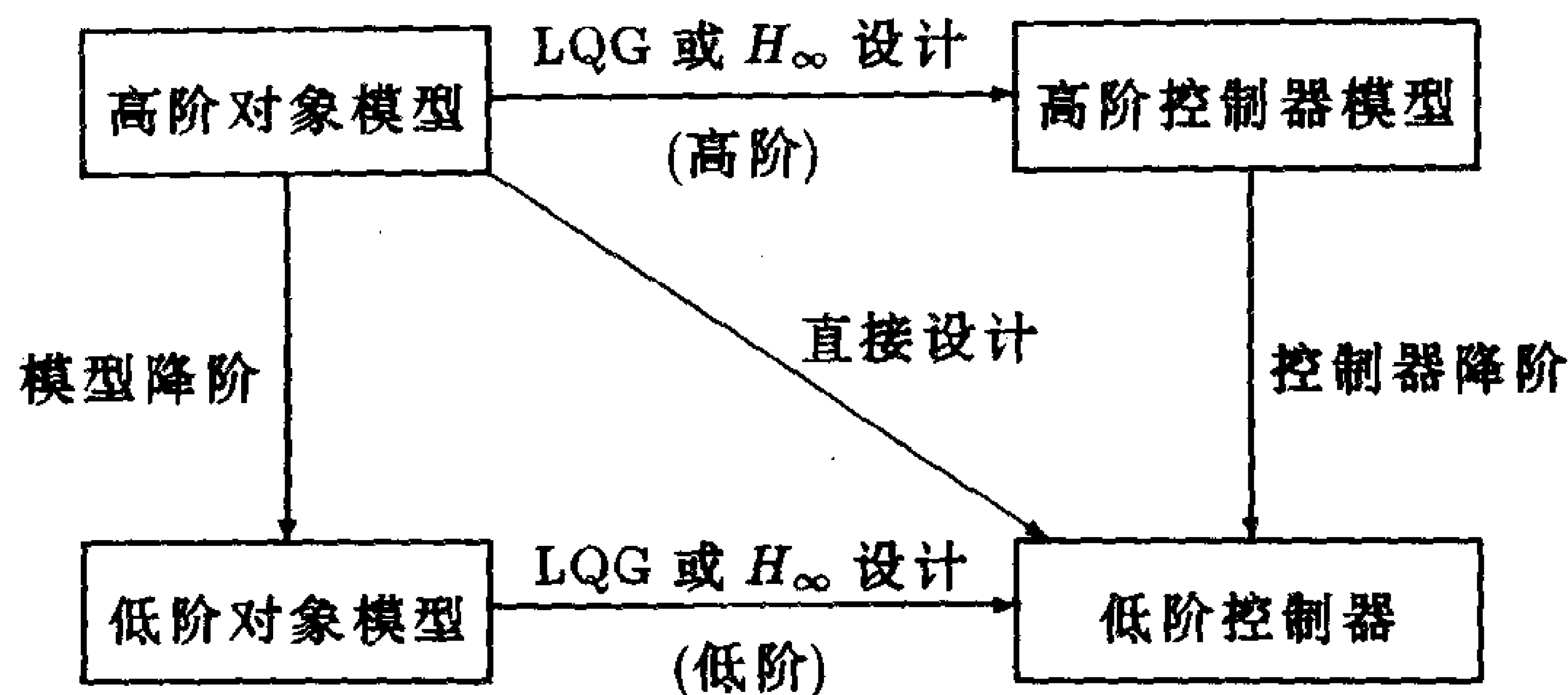


图7-30　低阶控制器设计框图

设计出低阶控制器来，最后再将控制器应用于原来的高阶对象模型的控制，其实很多控制器也是这样设计出来的，例如前面介绍 PID 自整定控制器时就建议利用这种途径来设计。图 7-30 中建议的第 3 种途径为对原系统设计一个全阶的 H_∞ 控制器，然后再对该控制器在闭环系统的基础上进行降阶处理，这一过程又常常称为控制器降阶 [2]，其实这样的方法是比较切实可行的，因为采用这样的方法除了考虑系统的开环特性之外又可以充分地考虑系统的闭环控制效果，得出一个能在控制效果上近似于高阶控制器的低阶控制器来。

7.6　时域设计方法的 MATLAB 工具箱

MATLAB 提供了多个可以直接使用的系统时域设计工具箱，如 John Little, Alan Laub 设计的控制系统工具箱 (control systems toolbox)， Richard Chiang 与 Machael Sofanov 编写的鲁棒控制工具箱 (robust control toolbox) 以及由 John Doyel 和 Keeth Glover 等人近期推出的 μ 分析与校正工具箱 (μ-analysis and synthesis toolbox) [4]，下面将分别介绍各个工具箱中的主要函数和功能。

7.6.1　控制系统工具箱简介

控制系统工具箱是美国著名学者 John Little 和 Alan Laub 推出的，它的出现是比较早的，正是由于出现了此工具箱，才使得 MATLAB 在国际控制界迅速流行起来。控制系统工具箱主要侧重于控制系统的计算机辅助分析，它当然也包括了其中一些经典的设计方法，如极点配置方法、线性二次型最优控制等。因为它包含了很多控制系统分析与设计的基本函数，所以在 CACSD 领域其它很多工具箱都要求在控制系统工具箱的支持下才能运行。控制系统工具箱的主要功能有

- **数学工具和方程求解**: 提供了常用的 Lyapunov 方程求解函数 `lyap()`，`dlyap()` 和 Riccati 方程的求解函数 `are()`，为求解控制问题奠定了较好的基础。
- **模型建立和转换**: 包括第 4 章中介绍的各种模型转换函数，如连续模型离散化函数 `c2d()`，传递函数与状态方程之间相互转换函数 `tf2ss()`，`ss2tf()` 等，以及模型实现

函数如相似变换函数 ss2ss(), 均衡实现函数 balreal(), 最小实现函数 minreal(), 及标准型变换函数 canon() 等。

- **频域和时域分析函数**: 包括阶跃响应分析函数 step(), 脉冲响应分析函数 impulse(), 频率特性分析函数如 bode(), nyquist(), nichols(), 根轨迹绘制函数 rlocus() 等。
- **控制系统设计函数**: 单变量 Ackermann 极点配置函数 acker(), 多变量系统极点配置函数 place(), 线性二次型最优控制函数 lqr(), 线性二次型估计器设计函数 lqe(), 以及这些函数的离散时间版本。

7.6.2 鲁棒控制工具箱简介

LQG 鲁棒控制和 H_∞ 最优控制理论提出之后，美国学者 Richard Chiang 与 Machael Sofanov 教授研制出了鲁棒控制工具箱，该工具箱主要侧重于控制系统的现代时域设计方法，当然该工具箱的调用方法并不如前面介绍的控制系统工具箱那么直观，因为涉及问题的复杂性大大地增加了，所以调用这些函数时需要一定的基础知识。鲁棒控制工具箱的基本内容简述如下：

- **系统模型建立与转换函数**: 在鲁棒控制工具箱下除了仍然可以使用状态方程和传递函数模型之外，为了调用方便还允许用它定义的数据结构模型来描述系统，如 mksys() 函数可以建立描述系统的树变量， augss() 和 augtf() 函数可以分别增广状态方程和传递函数模型，鲁棒控制工具箱还提供了模型转换的函数如多变量系统的双线性变换函数 bilin(), 传递函数矩阵到状态方程的转换函数 tfm2ss() 等。
- **多变量系统频域分析曲线绘制函数**: 如连续或离散时间系统特征增益曲线 cgloci(), dcgloci(), 奇异值 Bode 曲线绘制函数 sigma(), Perron 特征结构奇异值绘制函数 perron(), 结构奇异值 Bode 曲线绘制函数 ssv() 等。
- **模型降阶函数**: 如均衡实现的降阶方法 balmr(), obalreal(), Schur 相对误差降阶函数 bstschml() 和 bstschmr(), 最优 Hankel 范数近似降阶函数 ohklmr() 等。
- **鲁棒控制设计函数**: 如 LQG 控制器设计函数 h2lqg(), 及其离散时间版本 dh2lqg(), 连续时间 H_∞ 控制器设计函数 hinf() 及其离散版本 dhinf(), 采用 γ 的迭代 H_∞ 控制器直接设计函数 hinfopt(), 求取 $\|\cdot\|_2$ 的函数 normh2(), 求取 $\|\cdot\|_\infty$ 的函数 normhinf(), LQG 和 LTR 设计函数 lqg(), ltru(), ltry(), Youla 参数化公式求取函数 youla() 等。

鲁棒控制工具箱还提供了若干个演示例子可以直接使用，若想运行这些例子可以直接在 MATLAB 提示符下键入 rctdemo 字样，这将给出一个菜单驱动的演示程序，用户可以领略鲁棒控制工具箱提供的主要功能。

7.6.3 μ 分析与综合工具箱简介

在鲁棒控制工具箱出现之后， H_∞ 等技术又有了很大的发展，为了将最新的理论研究成果用 MATLAB 加以实现，就出现了称为 μ 分析和综合的工具箱，所谓 μ 分析即系



统的结构奇异值的分析，该工具箱是由包括 John Doyle 与 Keeth Glover 在内的众多滤波控制领域知名学者共同开发的，其主要内容包括：

- **模型转换函数**: 如将单变量模型转换为系统矩阵的函数 nd2sys()，由状态方程建立系统矩阵的函数 pck() 等及逆转换函数 unpck()，μ 工具箱还提供了自己的系统描述格式，并提供了由状态方程或传递函数转换成 μ 工具箱的标准描述格式的转换函数，如 pss2sys()，sys2pss()，zp2sys() 等。
- **模型降阶函数**: μ 工具箱也提供了模型降阶函数如最优 Hankel 范数近似降阶函数 hankmr()，互质系统均衡实现函数 sncfbal()，随机系统的均衡实现函数 srelbal()，系统矩阵的均衡实现函数 sysbal() 等。
- **H_2 与 H_∞ 分析与综合函数**: 连续与离散系统 H_∞ 最优设计函数 hfsyn() 和 dhfsyn()，H_∞ 的最小熵设计函数 hinfsyne()，连续系统的 H_2 最优时间函数 h2syn()，求取 $\|\cdot\|_2$ 的函数 h2norm()，求取 $\|\cdot\|_\infty$ 的函数 hinfnorm()，H_∞ 回路整形 (loopshaping) 设计函数 ncfsyn() 等。
- **μ 分析与综合函数**: D-K 迭代算法函数 dkit()，频率响应数据的传递函数拟合函数 fitsys() 等。
- **变化矩阵的分析与处理函数**: 如获得变化矩阵中的独立变量的函数 getiv()，独立变量排序函数 sortiv() 等，在这里就不加以介绍了。

习　　题

1) 已知线性 LTI 系统的状态方程模型参数 (A,B,C) 为

$$A=\begin{bmatrix}0&0&1.1320&0&-1\\0&-0.0538&-0.1712&0&0.0705\\0&0&0&1&0\\0&0.0485&0&-0.8556&-1.013\\0&-0.2909&0&1.0532&-0.6859\end{bmatrix},B=\begin{bmatrix}0&0&0\\-0.120&1&0\\0&0&0\\4.419&0&-1.665\\1.575&0&-0.0732\end{bmatrix},C=\begin{bmatrix}I_3&0_{3\times2}\end{bmatrix}$$

试将其中两个虚部绝对值最小的两个极点配置到 $-10,\ -5$。

2) 考虑下面给出的多变量系统传递函数矩阵

$$G(s)=\frac{1}{d(s)}\begin{bmatrix}29.20s+263.3&-(3.146s^3+32.67s^2+89.83s+31.81)\\5.679s^3+42.67s^2-68.84s-106.8&9.430s+15.15\end{bmatrix}$$

其中 $d(s)=s^4+11.67s^3+15.75s^2-88.31s+5.514$，试求出其解耦控制表示形式，并在解耦之后对其动态特性进行适当的校正。

3) 给出下面不稳定系统的解耦表示

$$G(s)=\begin{bmatrix}\dfrac{s+1}{s^2}&\dfrac{1}{s-1}\\\dfrac{s-1}{s^2}&\dfrac{1}{s-1}\end{bmatrix}$$

4) 若系统的状态方程模型为

$$\dot{x}(t)=\begin{bmatrix}0&1&0&0\\0&0&1&0\\-3&1&2&3\\2&1&0&0\end{bmatrix}x(t)+\begin{bmatrix}1&0\\2&1\\3&2\\4&3\end{bmatrix}u(t)$$

选择加权矩阵 $Q=\mathrm{diag}\{1,2,3,4\}$ 及 $R=I_2$，则设计出这一线性二次型指标的最优控制器及在最优控制下的闭环系统极点位置。

5) 对单变量系统 $G(s)=(s+4)/(s^4+6s^3+26s^2+46s+65)$ 进行 LQG 设计，并验证 $q=1$ (即不进行 LTR 补偿时)$u(t)$ 与 $\hat{}(t)$ 两处的返回比模型特征轨迹存在很大的差异，若想获得较好的 LTR 补偿则 q 应该取什么值？

6) 求出下列系统的 $\|G(s)\|_2$ 和 $\|G(s)\|_\infty$

$$G_1(s)=\frac{3s^3+13s^2+3s+10}{0.5s^4+1.5s^3+2s^2+s+1},\ G_2(s)=\frac{s^2+11s+10}{s^3+4s^2+9s+10},\ G_3(s)=\frac{1}{s^2+10^{-6}s+1}$$

7) 阅读并解释 $\|G(s)\|_\infty$ 求取函数 normhinf() 的主体部分清单

```
function nmhinf = normhinf(a, b, c, d, tol)
[rd,cd] = size(d); dtd = d'*d; hsv = hksv(a,b,c);
n_l = max([max(svd(d)),max(hsv)]);  n_h = max(svd(d)) + 2*sum(hsv);
while (n_h-n_l) > 2*tol*n_l
   gam = (n_l+n_h)/2; r = gam*gam*eye(cd) - dtd; ir = inv(r);
   Ham = [a+b*ir*d'*c  -b*ir*b'; c'*(eye(rd)+d*ir*d')*c -(a+b*ir*d'*c)'];
   lam = abs(real(eig(Ham))); ind = find(lam < sqrt(eps));
   [rind,cind] = size(ind);
   if rind == 0, n_h = gam;
   else, n_l = gam;  end
end
nmhinf = gam;
```

8) 试对下面的系统模型作互质分解

$$G_1(s)=\frac{5s+2}{s^2+5s+4},\ G_2(s)=\frac{s-1}{s(s-2)},\ G_3(s)=\frac{(s-1)^2(s^2-s+1)}{(s-2)^2(s+1)^3}$$

9) 给出单变量控制系统 $G(s)=(2s^2-1)/(s^5+2s^4+3s^3+5s^2+2s+2)$ 的最优 Hankel 降阶模型，对该模型作出适当的分析，并和第 4 章中介绍的降阶方法进行比较。

10) 给对象模型 $G(s)=\dfrac{(s-1)^2(s^2-s+1)}{(s-2)^2(s+1)^3}$ 设计一些镇定控制器，并比较性能。

参 考 文 献

[1] Anderson B D O. Controller design: moving from theory to practice (1992 Bode prize lecture. IEEE Control Systems Magazine, 1993, 13(4): 16-25

[2] Anderson B D O, Liu Y. Controller reduction: concepts and approaches. IEEE Trans. on Automatic Control, 1989, AC-34(8): 802-812

[3] Anderson B D O, Moore J B. Linear optimal control. Englewood Cliffs: Prentice-Hall, 1971

[4] Balas G, Doyel J, Glover K *et al.* μ-tools, the mu-analysis and synthesis toolbox. MUSYN Inc and the MathWorks, 1991

[5] Balasubramanian R. Continuous time controller design. IEE Control Engineering Series. Vol. 39. London: Peter Peregrinus Ltd, 1989

[6] Bass R W, Gura I. Higher order design via state space consideratins. Proceedings Joint American Control Conference. Troy, NY, USA, 1965, 311-318,

[7] Boyd S P, Balakrishan V, Kabamba P. A bisection method for computing the H_∞ norm of a transfer matrix and related problems. Mathematics in control signals and systems, 1989, 2: 207-219

[8] Chiang R Y, Sofanov M G. Robust control toolbox, user's manual. MathWorks, 1989

[9] Doyle J C, Francis B A, Tannerbaum A R. Feedback control theory. New York: MacMillan Publishing Company, 1991

[10] Doyle J C, Glover K, Khargoneckar P P, Francis B A. State space solutions to standard H_2 and H_∞ control problems. IEEE Trans. Automatic Control, 1989, AC-34(8): 831-847

[11] Erzberger H. Analysis and design of model following systems by state space techniques. Proceedings Joint American Control Conference, 1968

[12] Falb P L, Wolovich W A. Decoupling in the design and synthesis of multivariable control systems. IEEE Trans. Automatic Control, 1967, AC-12(6): 651-659

[13] Glover K. All optimal Hankel norm approximations of linear multivariable systems and their L_∞ error bounds. Int. J. Control, 1984, 39(6): 1115-1193.

[14] Glover K, Doyel J C. State space formulae for all stabilizing controllers that satisfies an H_∞ morm bound and relations to risk sensitivity. Systems and Control Letters, 1988, 11: 167-172

[15] Grimble M J. LQG optimal control design for uncertain systems. Proceedings IEE, Pt D, 1990, 139: 21-30

[16] Kautsky, Nichols, Van Dooren. robust pole assignment in linear state feedback. Int. J. Control, 1985, 41: 1129-1155

[17] Kwakernaak H, Robust control and H_∞-optimisation — tutorial paper. Automatica, 1993, 29: 255-273

[18] MacFarlane A G J, Scott-Jones D F A. Vector gain. Int. J. Control, 1979, 29: 65-91

[19] Maciejowski J M. Multivariable feedback design. Wokingham: Addison-Wesley, 1989

[20] Mufti I H. A note on the decoupling of multivariable systems. IEEE Trans. Automatic Control, 1969, AC-14(4): 415-416

[21] Patel R V, Munro N. Multivariable system theory and design. Oxford: Pergamon Press, 1982

[22] Sofanov M G, Jonckheere E A, Verma M, Limebeer D J N. Synthesis of positive real multivariable feedback systems. Int. J. Control, 1987, 45: 817-842

[23] Stein G, Athens M. The LQG/LTR procedure for multivariable feedback control design. IEEE Trans. on Automatic Control, 1987, AC-32(2): 105-114

[24] Wolovich W A. Linear multivariable systems. New York: Springer-Verlag, 1974

[25] Zames G. Feedback and optimal sensitivity: model reference transformations, multiplicative seminorms, and approximate inverses. Proceedings 17th Allerton Conference, 1979, 744-752

第 8 章 MATLAB 下的图形界面设计技术与应用

8.1 MATLAB 图形界面概述

早期的 MATLAB 版本只提供了一个命令屏幕和一个图形屏幕，用户只能在这样两个屏幕之间作切换，如果用户在命令屏幕上给出一条 MATLAB 绘图命令，则要想得出相应的图形，MATLAB 会自动地切换到图形屏幕上，如果再给出一条命令，则会自动地切换到原来的命令屏幕，虽然 MATLAB 可以在两个屏幕之间作自动的切换，但使用起来还是极其不便的，尤其想同时显示出多种曲线时更是如此。此外，MATLAB 早期的版本并不适于开发用户友好的图形界面程序。

随着多窗口 (Windows) 技术的发展，MATLAB 的用户及 MathWorks 公司的开发者们逐渐意识到在多个窗口界面下运行 MATLAB 的必要性和可行性。1992 年 MathWorks 公司推出了具有创造性意义的 MATLAB 4.0 版本，并于次年 (1993 年) 正式推出了 MATLAB 4.0 版的 PC 机版本，以配合日益流行的 Microsoft Windows 一起使用。MATLAB 4.0 版本一出现，立即引起了使用者和程序开发人员的极大兴趣，因为它使得在其它语言环境下看起来十分复杂的 Windows 图形界面设计显得非常的容易和方便。

MATLAB 4.1 及 4.2 版本的出现更为图形界面的设计提供了新方法，比如在该方法下提供了标准的联机帮助系统的设计方法，并提供了若干个标准的对话框可以直接调用，在对话框控制元件上又增加了新的内容。本书将基于 MATLAB 4.0 的图形界面设计技术作一下介绍，熟悉该风格之后也利于充分使用高版本的新功能。

8.2 图形窗口的设置

在 MATLAB 4.0 下，如果用户想打开一个新的图形窗口，则可以选择 MATLAB 命令窗口中的 File | New 菜单下的 Figure 子菜单，这样将获得一个标准的 MATLAB 图形窗口，当然如果采用下面的命令将使得打开窗口的形式更富于变化

```
窗口句柄=figure(属性 1，属性值 1，属性 2，属性值 2，···)
```

其中的属性和属性值在后面介绍 set() 函数时再详细介绍。显然，用户可以通过这样的方式很容易地打开一个新的图形窗口，并返回该窗口的句柄，使得以后能够简单地对该窗口的属性进行进一步的修正。建立起来窗口 (亦即成功地获得窗口句柄) 之后，用户还可以调用 figure() 函数来显示该窗口，并将之设定为当前的窗口，这时该函数具体的调用格式为



```
figure(窗口句柄)
```

其实，即使这里引用的窗口句柄不存在，也可以使用这一命令，它的作用是对这一窗口句柄生成一个新的窗口，并将之定义为当前窗口。

当然在 MATLAB 环境下允许用户同时打开很多的窗口，而每一个窗口应该对应于自己的句柄 [1)]，用户可以调用 gcf() 函数来获得当前窗口的句柄，以便对之进行下一步的操作。

在第 2 章中已经概略地介绍过，如果用户想改变图形窗口的一些属性，则可以通过调用 MATLAB 提供的 set() 函数来完成，前面介绍过， set() 函数的调用格式为

```
set(窗口句柄, 属性 1, 属性值 1, 属性 2, 属性值2,···)
```

其中用户可以利用窗口句柄来指明要处理的窗口。因为当前窗口句柄可以由 gcf() 函数来获得，所以如果给出 MATLAB 命令

```
set(gcf, 'Color',[1,1,1])
```

则可以将当前窗口的颜色变为白色。除了窗口的背景颜色设置之外， MATLAB 还允许对其它各种图形窗口的属性作修正，其中图形窗口的常用属性如下：

- Color **属性**：可以设置图形窗口的界面颜色，其属性值可以为红绿蓝三原色的不同配比构成的 1×3 向量构成，其中每个向量分量的取值范围为 0 到 1，这样就可以获得各式各样的颜色变化。如果这些颜色配比值取作极端值 0 或 1 时，则可以得出八种颜色组合，如表 8-1 所示。

表8-1 各种常用的颜色配比

配比向量	颜　色	配比向量	颜　色	配比向量	颜　色	配比向量	颜　色
[0 0 0]	黑色	[1 1 1]	白色	[0 0 1]	蓝色	[0 1 0]	绿色
[0 1 1]	天蓝色	[1 0 0]	红色	[1 0 1]	粉色	[1 1 0]	黄色

在 Windows 环境的颜色分辨率设置得较高时，也可以将各个颜色配比的值设置成 0 到 1 之间的任意小数，从而使得整个颜色系统显得更加美观。在 MATLAB 下默认的窗口背景颜色为黑色，而执行 whitebg() 函数之后将自动变成白色。

- InvertHardcopy **属性**：此属性用来控制是否在图形打印时作颜色逆转处理，它的属性值可以选择为 'on' 和 'off'，在选择为 'on' 时，则将对打印机输出的格式作逆转处理，亦即如果原来的图形窗口底色为黑色时，将在打印机上以白色的形式表示出来，而原来设置成白色的颜色将在打印机上以黑色的形式打印出来，这一选项是默认的。该属性值取作 'off' 时则不作这样的颜色逆转，在图形窗口上有什么颜色的图形就将用什么颜色打印出来（当然在单色打印机上只能由不同的灰度来表示）。

[1)] 句柄是计算机编程中的一个常用的概念，形象地说，每一个窗口的句柄相当于一个窗口的名字或标志，它使得计算机能够方便地从众多窗口中找出所要操作的窗口。

- **MenuBar 属性**：设置图形窗口菜单条形式，可为 'none' (不加菜单条) 或 'figure'(图形默认) 两个选项，用户如果选中了 'none' 选项值，则在当前处理的窗口内将没有菜单条，这时用户可以根据后面将介绍的 uimenu() 函数来加入自己的菜单条，如果用户选择了其中的 'figure' 选项值，则该窗口将保持图形窗口默认的菜单项，其中相应的选项后面将进行介绍，这一设置为默认的。就是选择了 'figure' 选项也还可以采用 uimenu() 函数在原默认的图形菜单后面添加新的菜单项。
- **Name 属性**: 设置图形窗口的标题栏内容，它的属性值应该是一个字符串，在图形窗口的标题栏中将把该字符串内容填写上去。
- **NumberTitle 属性**: 决定是否设置图形窗口标题栏的图形标号，它相应的属性值可选为 'on' (加图形标号) 或 'off' (不加标号)，若选择了 'on' 选项则会自动地给每一个图形窗口标题栏内加一个 Figure No *: 字样的编号，即使该图形窗口有自己的标题也同样要在前面冠一个编号，这是 MATLAB 的默认选项，若选择 'none' 选项则不再给窗口标题进行编号显示了。
- **Position 属性**: 用来设定该图形窗口的位置和大小，其属性值是由 4 个元素构成的 1×4 向量，其中前面两个值分别为窗口左上角的横纵坐标值，后面两个值分别为窗口的宽度和高度，其默认的单位为像素点 ('pixels')。
- **Units 属性**：除了前面用到的像素点单位之外，还允许用户使用一些其它的单位，如 'inches' (英寸)、'centimeters' (厘米)、'normalized' (归一值，即 0 和 1 之间的小数) 等，这种设定将影响到一切定义大小的属性项，如前面介绍的 Position 属性。
- **Resize 属性**：用来确定是否可以用鼠标器来调整所建立起来的图形窗口的大小，这里当然有两个参数可以使用 'on' (可以调整) 和 'off' (不能调整)，其中的 'on' 选项为默认的选项。
- **Pointer 属性**：用来设置在该窗口下指示鼠标器位置的光标表示的显示形式，它的可选值及对应的表示形式如表 8-2 所示，其中的 'arrow' 选项为默认的形式。

表8-2 Pointer取值及光标形式示意图

选 项	意 义	选 项	意 义	选 项	意 义
crosshair	细十字花型	arrow	箭头指示	watch	砂漏指示 (表示等待)
topl	左下到右上角箭头	topr	左上到右下箭头	botl	形同 topr, 意义不同
botr	形同 topl, 意义不同	circle	圆形光标指示	cross	双线十字花型
fleur	带箭头的十字花				

- **Visible 属性**：它用来决定建立起来的窗口是否初始时刻处于可见的状态，它的选项为 'on' 和 'off' 两种，其中 'on' 选项为 MATLAB 的默认状态。如果选择了 'on' 选项，则窗口改变大小 (包括从默认的大小改变到用户设定的大小)、颜色等一切的过程均是可见的，所以有时在实际编程中并不希望采用这样的方式，而希望把动态改

变窗口形式的过程都给隐含起来，这就需要首先选择 'off' 选项，在更改完成之后再将该选项设置为 'on' 来显示窗口的最终形式。

- UserData 属性：用户可以通过 UserData 提供的数据区域进行自己数据的传递，比如用户可以用 set() 函数给某一句柄附加地设置一些相关的参数，这些参数是以一个矩阵来表示的，如果想使用这样的矩阵，则可以再由 get() 函数调用出来。例如若用户想让 A= [1,2,3; 4,5,6; 7,8,9] 及 B= [1,3,5] 两个矩阵随着窗口句柄 gwin 而传递，则可以给出下面的命令

```
>>  A=[1,2,3; 4,5,6; 7,8,9]; B=[1,3,5];
>>  set(gwin, 'UserData',[A; B]);
```

这时会自动将 A 和 B 写成一个矩阵的形式设置到 gwin 句柄上作为用户数据，用户可以通过这样的方法给句柄添加上一些附加参数，使得它们可以随着句柄而进行自动的传递，可见这一属性在实际编程中是相当实用的。这样若再想取出 A 和 B 矩阵，则可以给出下面的命令

```
>>  AA=get(gwin, 'UserData')
AA =    1     2     3
        4     5     6
        7     8     9
        1     3     5
>>  A=AA(1:3,:); B=AA(4,:);
```

- 在窗口属性中有一些是对打印纸进行设置的，包括 PaperUnits (打印纸长度单位，默认值为 'inches')、PaperOrientation (打印走向，有两种选项 'portrait' 和 'landscape'，分别表示纵向和横向打印，其中前者是默认的走向)、PaperPosition (打印纸的位置)、PaperSize (打印纸的大小，是一个 1×2 的向量，分别存储纸张的宽度和长度)、PaperType (打印纸类型) 等，这些参数可以由 set() 函数来设定，也可以由图形窗口的 File | Print Setup 菜单来设置，该菜单将给出一个如图 8-1 所示的对话框，可见可以在该对话框中方便地设置出上面的各个打印机控制参数。
- 除了上述的属性之外，MATLAB 还允许对键盘和鼠标器键按下这样的动作进行响应，这类属性有 KeyPressFcn (键盘键按下响应)、WindowButtonDownFcn (鼠标键按下响应)、WindowButtonMotionFcn (鼠标器移动响应) 及 WindowButtonUpFcn (鼠标键释放响应) 等，这些属性所对应的属性值可以为用 MATLAB 编写的函数名或命令名，表示一旦指定的键盘或鼠标键按下之后，将自动调用属性值，即给出的函数或命令。

注意，在使用 set() 命令时，每个属性项的名称应该用单引号括起来，而其后面的属性值如果为字符串，则也应该用单引号括起来。

采用 get() 函数可以得出窗口或其它句柄的有关信息，该函数的调用格式为

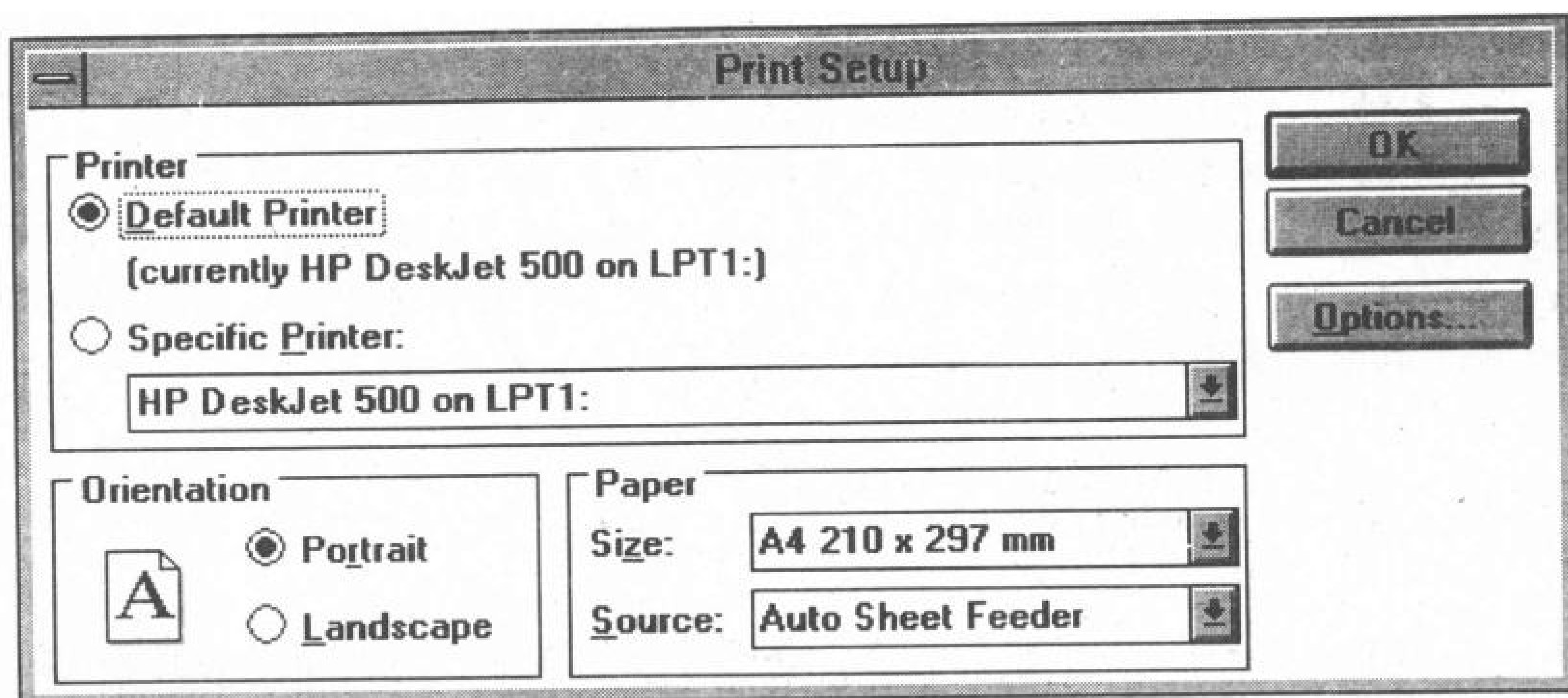


图8-1 图形窗口打印机参数设置对话框

V=get(句柄名，属性)

其中 V 为指定属性的返回属性值。例如若想测试当前窗口的颜色，则可以给出命令

V=get(gcf，'Color')

这样返回的 V 为当前窗口颜色向量。如果在 get() 函数调用时不给出任何的属性名，则将返回当前句柄所对应的所有属性值。例如若采用 get(gcf) 命令则将获得以下的查询结果[1)]

```
>>  get(gcf)
   BackingStore = on                        ]
   Color = [0 0 0]                          InvertHardcopy = on
   Colormap = [ (64 by 3) ]                 KeyPressFcn =
   CurrentAxes = [0.000610352]              MenuBar = figure
   CurrentCharacter =                       MinColormap = [64]
   CurrentMenu = [1]                        Name =
   CurrentObject = []                       NextPlot = add
   CurrentPoint = [0 0]                     NumberTitle = on
   FixedColors = [                          PaperUnits = inches
           0 0 0                            PaperOrientation = portrait
           1 1 1                            PaperPosition = [0.25 2.5 8 6]
           1 1 0                            PaperSize = [8.5 11]
           1 0 1                            PaperType = usletter
           0 1 1                            Pointer = arrow
           1 0 0                            Position = [120 120 560 420]
           0 1 0                            Resize = on
           0 0 1                            SelectionType = normal
```

1)因版面所限，这里由双列的形式列出有关信息。

```
ShareColors = yes                Children = [0.000610352]
Units = pixels                   Clipping = on
WindowButtonDownFcn =            Interruptible = no
WindowButtonMotionFcn =          Parent = [0]
WindowButtonUpFcn =              UserData = []
ButtonDownFcn =                  Visible = on
```

可以看出，这里的许多属性值都是空的，例如 Name 属性对应项有 Name= 字样，这表示当前的窗口没有自己的标题，同时 UserData 后面跟的附加矩阵参数也是空的，表示没有数据随着此句柄一同传递。

值得指出的是， get() 函数并不限于获得窗口句柄的有关信息，它还可以获得其它句柄所对应的信息，例如用户用 get(gca) 命令可以测试一下当前坐标轴设置的有关信息，这里 gca() 函数的格式和 gcf() 相似，是用来获得当前坐标轴句柄的。在后面的介绍中还将介绍如何采用 get() 函数来对对话框控制元件进行操作。

为了方便起见，MATLAB 还专门提供了设置白色图形背景窗口的函数 whitebg()，它除了将原窗口的背景颜色设置成白色之外，还对其它一些参数，如前景颜色、坐标轴颜色及打印时颜色逆转等参数，都作了相应的设置，比如坐标轴的颜色从原来的白色变成了黑色，以保证坐标轴可以充分地显示出来。

例 8.1　如果用户想打开一个新的图形窗口，则可以给出如下的命令

```
gwin=figure('Visible','off');
set(gwin, 'Color',[1,0,0], 'Position',[100,200,300,400],...
   'Name','My Own Program','NumberTitle','off', 'MenuBar','none', ...
   'KeyPressFcn','disp(''Hello, Keyboard key pressed'')');
set(gwin, 'Visible','on')
```

这一程序段将首先定义一个不可见的窗口，并将其句柄设置为 gwin，然后采用 set() 函数的 'Color' 属性将它的背景颜色设置为红色，并用 'Position' 属性将其大小和位置设置出来，再给该窗口设置一个标题，最后将取消原来窗口的菜单条。在这一程序段中还定义了对 KeyPressFcn 属性的响应函数，即在用户按下任意一个键盘键时将调用 disp() 函数来显示 Hello, Keyboard key pressed 字样。这里的一切设置完成之后，则可以再调用 set() 函数将该窗口显示出来。注意，对键盘键按下这一事件的响应仅限于当前的图形窗口为活动窗口时才起作用。对这样设计的窗口若使用 get(gwin, 'Position') 和 get(gwin, 'Color') 命令将分别返回下面的结果

```
>>  get(gwin, 'Position')
ans = 100    159    300    400
>>  get(gwin, 'Color')
ans =   1     0     0
```

可以看出，使用 set() 命令可以很容易地设置或修改 Windows 界面的属性，并且马上可以得出修改的结果，而不需像 C 语言那样在编写程序之后还得去对之进行编译连接调试。用户当然还可以写出下面貌似更简单的程序段

```
gwin=figure;
set(gwin, 'Color',[1,0,0], 'Position',[100,200,300,400],...
   'Name','My Own Program','NumberTitle','off', 'MenuBar','none');
```

用户可以在实际运行时比较一下这两组命令所产生的结果。从效果上看，这两个程序段都将打开同样的图形窗口，并同样将该窗口的句柄赋给 gwin 变量，但在后一个程序段中，窗口显示的修改过程都将在计算机上显示出来，而前一个程序段隐含起来了所有的中间过程，仿佛直接打开了一个按照要求选定的新窗口一样，所以在实际编程时往往采用前一种方法。事实上还可以由

```
gwin=figure('Color',[1,0,0], 'Position',[100,200,300,400],...
   'Name','My Own Program','NumberTitle','off', 'MenuBar','none');
```

来取代上面的语句，其效果是完全一致的。可见，这一种方法调用更加简单。

MATLAB 允许用户使用 SWin=get(0,'ScreenSize') 来测取当前 Windows 环境的屏幕分辨率，这一命令将返回一个 1×4 的向量 SWin，其中前两个分量分别为屏幕的左下角横纵坐标 (取作 0,0)，而后两个分量分别为屏幕的宽度和高度，例如 Windows 环境的整体设置为 800×600 分辨率，则调用此函数将返回

```
>> SWin=get(0,'ScreenSize')
SWin = 0      0  800  600
```

这样的结果。这时用户就可以依照现行屏幕的分辨率来设置窗口的大小，例如若想建立一个起始于屏幕左下角，宽度和高度均为现行窗口分辨率大小一半的图形窗口，则可以给出如下的命令

```
set(gwin, 'Position',[0,0,0.5*SWin(3),0.5*SWin(4)])
```

这样将有助于用户编写出更通用的软件以适应不同屏幕类型及分辨率的特殊要求。除了采用默认的 'pixels' 单位制来进行赋值以外，比较好的单位制选取应该为 'normalised' (或简写为 'normal')，例如前面的语句可以写成

```
set(gwin, 'Position',[0,0,0.5,0.5], 'Units', 'normal')
```

8.3 菜单环境的使用与创建

8.3.1 标准 MATLAB 菜单及使用

MATLAB 4.0 和以前版本的重要区别在于它除了传统的命令输入方式以外，还提供了方便的菜单提示及输入方法，回顾图 2-2 中给出的 MATLAB 图形界面可以看出，MATLAB 的命令窗口有自己的菜单条，在第 2 章中曾分别介绍了其中的一些常用的菜单项，使用了这些菜单项之后就可以用一种更简洁的方式来操作 MATLAB 了。由于 MATLAB 允许用户打开多个不同的窗口，所以它给出的 Window 菜单中列举了已经打开的各个窗口名称的表格，用户可以从中选择一个要切换到的窗口名，从而轻易地切换到指定的窗口。

MATLAB 的各个图形窗口也有自己的菜单条，其中两个主要菜单的内容分别如图 8-2(a), (b) 所示， 其中的 File | New Figure 将允许自动打开一个新的图形窗口，而 File | New 菜单项将给用户一些其它的选择，如 M-File (打开一个新的 M 文件)， Figure (打

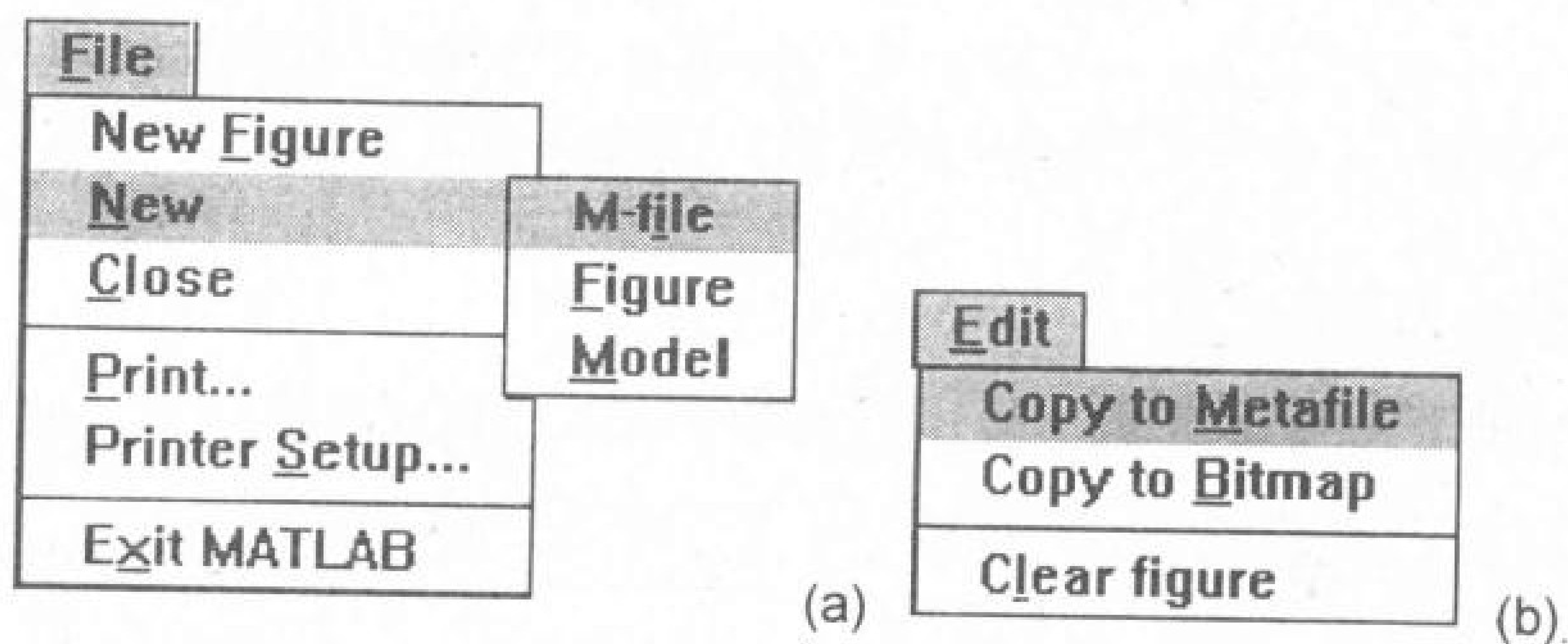


图8-2 MATLAB图形窗口的菜单项

开一个新的图形窗口）及 Model（打开一个新的 SIMULINK 编辑界面）。File | Close 将允许用户关闭当前的图形窗口，File | Print 将调出一个如图 8-3 所示的对话框，来询问用户打印内容，选择完成之后将会自动地将图形窗口的内容送往打印机去打印。

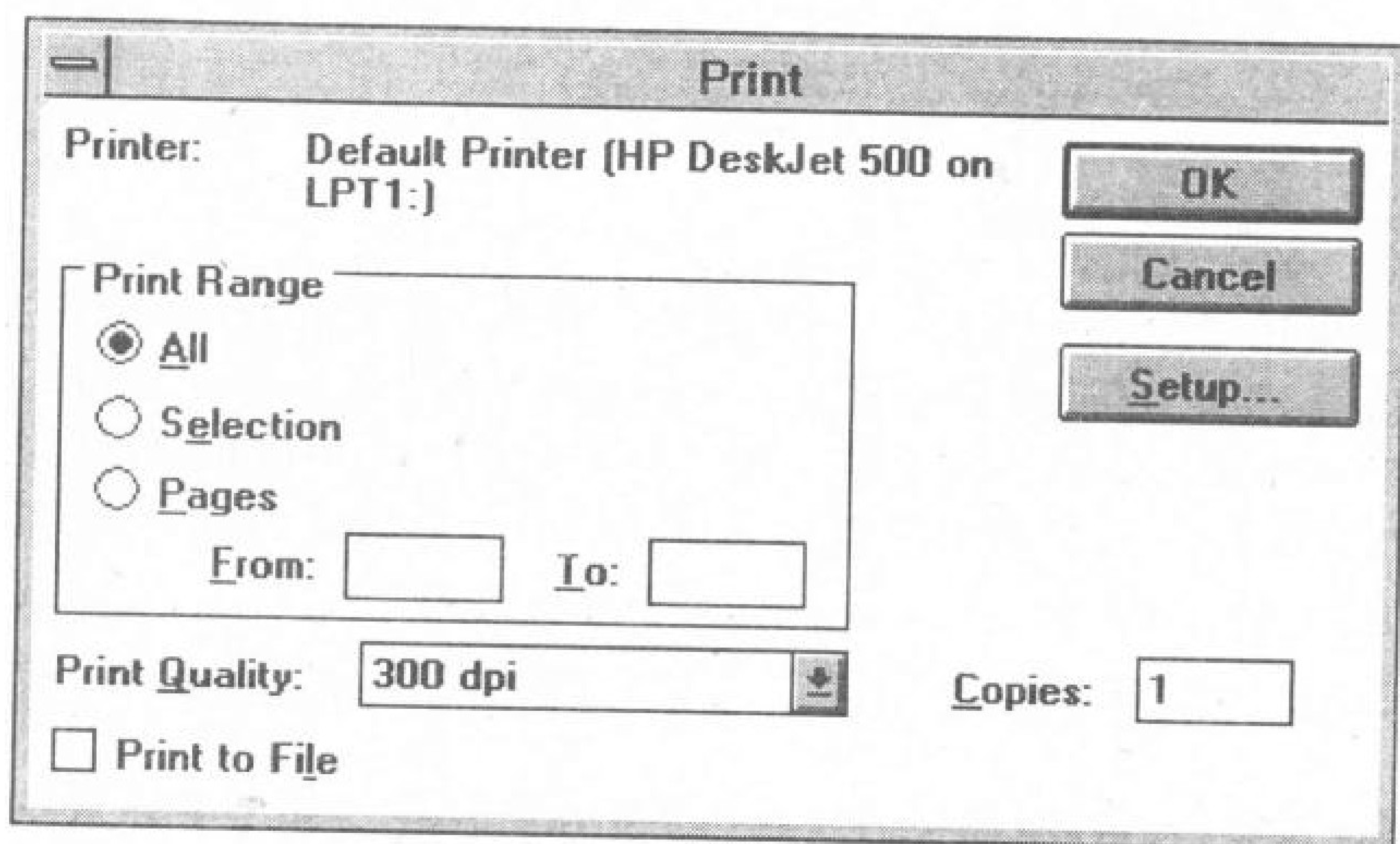


图8-3 图形打印参数对话框

MATLAB 的 Edit 菜单中有 3 个选项，其中 Clear Figure 显然用来对当前的图形窗口进行清屏，这一命令的作用和 MATLAB 命令 clg 是完全一致的。Edit | Copy to Metafile（复制成图元）菜单选项将窗口中得出的图形按照图元文件的格式存放到剪切板中，Edit | Copy to Bitmap（复制成位图）菜单选项将窗口中的图形按照位图文件的格式存放到剪切板中，这两种文件的格式是有区别的，其中图元文件是矢量化的，而位图文件并不是，所以说，一般将结果存储为图元文件的格式更方便，因为这样得出的图形在放大和缩小时会自动地按照指定的比例重新绘制图形，而不是像位图文件那样对各个像素点进行处理．所以采用图元文件格式在放大和缩小时能使图形保持原样的。位图存放规则并不适合于高质量的绘图，且一般情况下占用的存储空间较大，所以一般最好采用图元的格式来存储图形。

和MATLAB命令窗口一样，图形窗口的Window菜单中列出了所有从属于MATLAB的窗口名称，这允许在MATLAB下自由地切换窗口，而Help菜单给出MATLAB各个函数的调用方法，而不是针对图形窗口的使用方法给出帮助信息。

8.3.2 简易菜单系统的设计

MATLAB提供了简易菜单系统的设计函数menu()，该函数的调用格式为

```
k=menu 标题，选项 1，选项 2，选项 3,···);
```

早期MATLAB版本将给出编号的菜单提示，用户可以给该菜单输入相应的编号即可，例如

```
k=menu('Choose a color','Red','Green','Blue');
```

命令将给出下面的菜单显示

```
----- Choose a color -----
      1) Red
      2) Green
      3) Blue
Select a menu number:
```

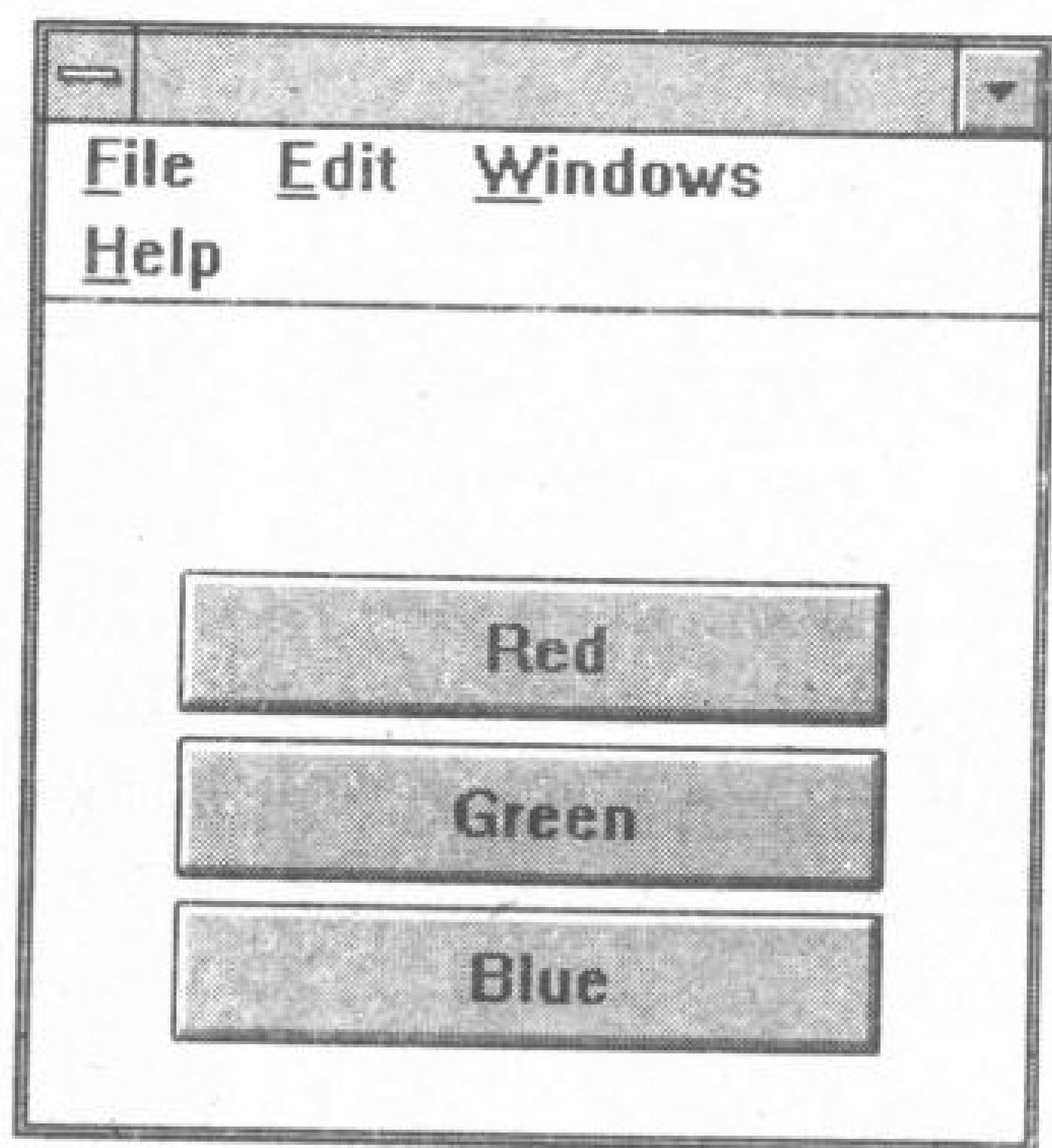
用户可以根据提示输入1,2, 或3来响应此菜单，输入完成之后此函数会自动地将此数值传送给k变量(即使传送的数值不在给定的范围内也会接受此值)，在MATLAB 4.0及以上版本中对此函数的显示格式作了一些修正，例如上面的命令将给出如图8-4 (a)所示的菜单窗口，允许用户用鼠标来选择其中的菜单项。另外MATLAB还提供了choices()函数来生成菜单系统，该函数的调用格式为

```
choices( 标题，菜单名称，提示串，回调函数列表)
```

其中标题是写到窗口的标题栏中的字符串(用引号括起)，菜单名称是显示在菜单上的字符串，提示串是以字符串构成的矩阵，其中每个子字符串将出现在菜单按钮的上面作为提示文字，后面的回调函数列表中将列出对提示串中各个菜单项响应所需的函数名构成的矩阵，其长度应该和提示串一致，这时在某一个菜单项被选中后， MATLAB将自动在回调函数列表中选择出对应的回调函数，并启动该函数来响应此菜单。此函数和menu()的区别在于，在menu()函数下一个菜单项被选中，则会自动关闭菜单窗口，而choices()函数在回调函数执行完成后还会重新显示对应的菜单，允许用户作其它菜单的选择，在choices()函数生成的菜单中将有一项标注为Close, 选中了此选项将会自动退出本菜单系统。MATLAB提供的一套演示程序demo.m就是采用这样的驱动方式来编写的，其中该函数的主要内容为[1)]

```
labels = str2mat('Introduction','MathWorks Specials','Data Analysis',...
```

1)由于版面需要，在格式上对此程序作了相应的改写。



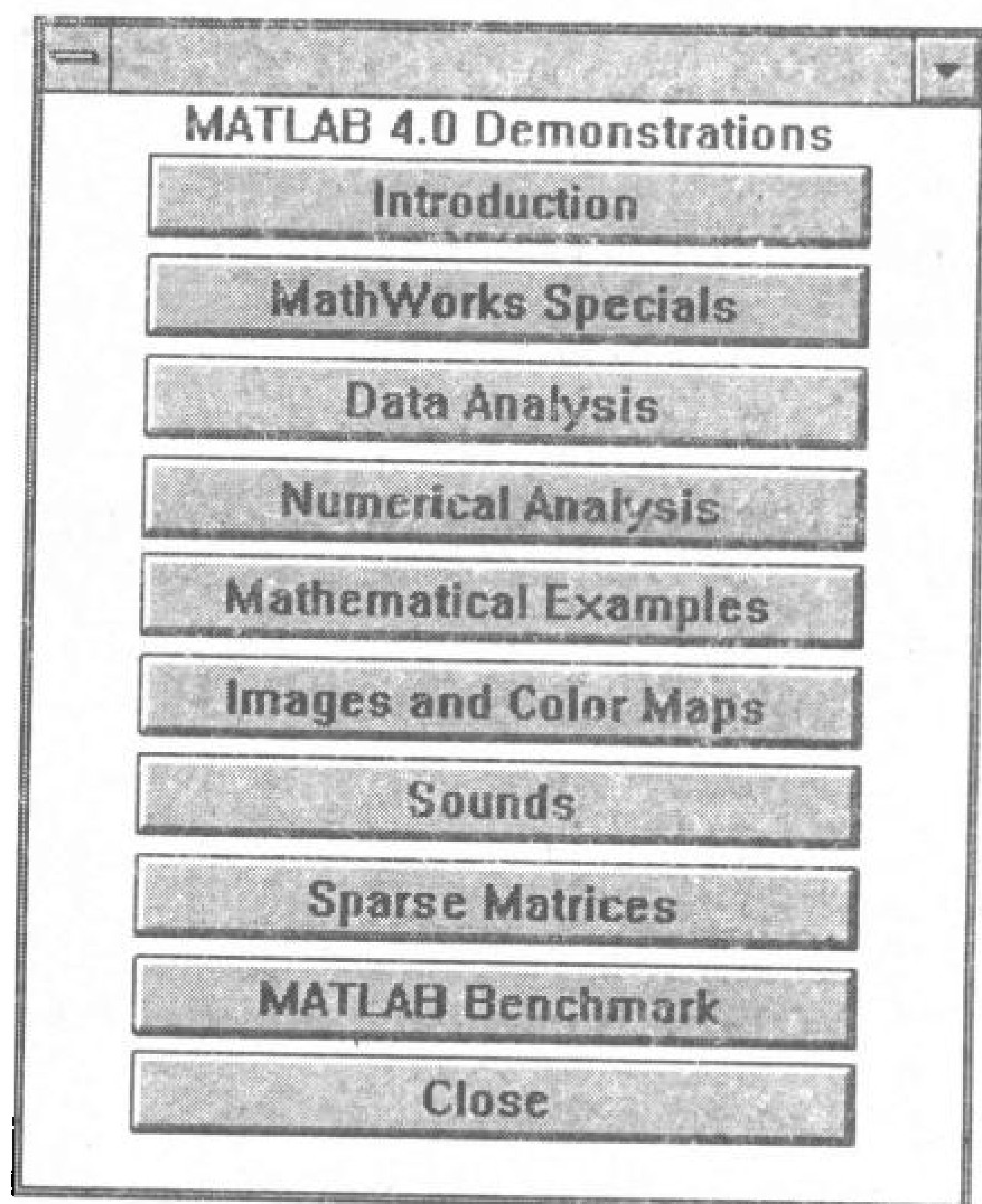


图8-4 简易菜单系统窗口及演示

```
   'Numerical Analysis','Mathematical Examples','Images and Color Maps',...
   'Sounds','Sparse Matrices','MATLAB Benchmark');
callbacks = ['intro          '; 'specials      '; 'datalist      '
             'nalist         '; 'mathlist      '; 'imagedemo     '
             'sounddemo      '; 'sparlist      '  'bench         '];
choices('DEMO', 'MATLAB 4.0 Demonstrations', labels, callbacks);
```

这里调用了 str2mat() 函数将字符串的列表构成一个规整的字符串矩阵，其调用格式是显然的，用户可以由联机帮助系统获得更详细的信息。上面一组命令将给出一个如图8-4(b) 所示的菜单界面。若用户选择了其中某一个菜单项后，还允许再进一步调用另外的 choices() 函数，所以通过这样的方式用户可以建立起自己的菜单系统。

8.3.3 用户自定义菜单的设计与使用

有 Windows 使用经验的用户会很自然地体会到 Windows 程序界面和其它程序的一个重大的区别在于 Windows 程序一般有较美观的操作环境(即界面)，例如它有意义十分显然的菜单提示和对话框显示，这样用户往往在没有说明书的情况下也能较好地使用该应用程序。

MATLAB 除了提供标准的菜单之外，还允许用户设计出自己所需要的菜单系统，其菜单设置是由 uimenu() 函数来完成的，该函数的调用格式为

```
菜单项句柄 =uimenu(窗口句柄, 属性 1, 属性值 1, 属性 2, 属性值 2, ···)
```

注意，这里的属性名称应该用单引号括起来。各个属性的名称和取值内容将在后面陆续介绍。如果想在某一个菜单条下定义一个新的子菜单条，则可以使用下面给出的命令格式

```
子菜单句柄 =uimenu(菜单项句柄, 属性 1, 属性值 1, 属性 2, 属性值 2, ···);
```

其中 uimenu() 函数的属性有多种，但最基本的必须有以下两条：

- **菜单条的提示名称** (Label)：可以是一个任意字符串。和编写 C 语言的资源文件一样，在菜单项提示的字符串中允许用户使用 & 标志，以表示该符号后面的字符在显示时有一个下划线修饰，这使得用户可以用键盘键容易地激活相应的菜单项。
- **回调函数** (Callback)：可以为一个函数名称 (用引号括起)，也可以是一组 MATLAB 命令，在该菜单条被选中以后，MATLAB 将自动地调用此回调函数来作出对相应菜单项的响应，如果没有设置一个合适的回调函数，则此菜单项也将失去其应有的意义。

其中在产生下拉式菜单时 Callback 选项也可以省略，因为这时可以直接打开下一级菜单，而不是侧重于对某一函数进行响应，除了上面两个属性之外，其它较常用的菜单属性可以归纳如下：

- **热键名称定义** (Accelerator)：用户可以设置相应于菜单条的热键命令，如写成 Alt+F2 则表示热键定义为 *Alt*+F2 键。
- **背景颜色与前景颜色** (BackgroundColor 和 ForegroundColor)：均为 1×3 的向量，它们可以用来描述前景颜色和背景颜色，该向量的三个分量分别表示红绿蓝三元色的分量配比，而每个颜色分量的配比值应该在 0 和 1 之间，例如 [0,0,0] 表示白色，[1,0,0] 表示红色，[0,1,0] 表示绿色等。在 Microsoft Windows 环境的颜色分辨率设置的比较高 (如 256 色) 时，这三个颜色分量还可以取为 0 到 1 之间的小数以丰富图形界面颜色的选择。
- **初值是否为已选状态** (Checked)：它的取值可为 'on' 和 'off' 两种，其中的 'off' 为默认的状态。
- **使能状态** (Enable)：其取值可为 'on' 和 'off' 两种，其中的 'on' 为默认的状态，表示该菜单项处于可选状态，若其值为 'off' 则表示该菜单项处于禁选状态。
- **菜单条的位置** (Position) 可以用来设置菜单项的相对排列位置，可以用来测试该菜单处于整个菜单系统的第几位。
- **分界符** (Separator)：其取值可为 'on' 和 'off' 两种，表示在该菜单项的上面是否加一个分界符 (即一条横线)，如果不指定该选项，则其默认状态对应于 'off'。

例 8.2 从上面的叙述中可见，利用 MATLAB 提供的图形界面设计函数可以很容易地建立起一个菜单系统，例如使用下面命令



```
screen=get(0,'ScreenSize');
WinW=screen(3); WinH=screen(4);
gmain=figure('Color', [0,1,1], 'Pos', [0, 0, 0.4*WinW, 0.3*WinH], ...
   'Name','My New Window', 'NumberTitle', 'off', 'MenuBar', 'none');
mfile=uimenu(gmain, 'label','&File');
mhelp=uimenu (gmain, 'label','&Help');
uimenu(mfile, 'label', '&New', 'call','disp(''New Item Selected'')');
uimenu(mfile, 'label', '&Open', 'call','disp(''Open Item Selected'')');
msav=uimenu(mfile, 'label', '&Save', 'Enable','off');
uimenu(msav, 'label', 'ASCII file', 'call','key=3;k0=1; filesys;');
uimenu(msav, 'label', 'Graphics file', 'call','key=3; k0=2; filesys;');
uimenu(mfile, 'label', 'Save &As', 'call', 'key=4; filesys; ');
uimenu(mfile, 'label', '&Exit', 'separator', 'on','call', 'close (gmain)');
uimenu(mhelp, 'label', 'About... ', 'call', ...
   ['disp(''my help''); set(msav, ''Enable'',''on'') ']);
```

这一段程序首先测试出整个屏幕的分辨率，并将窗口的宽度和高度赋给 WinW 和 WinH 变量，然后在屏幕的左下角处打开一个宽度和高度分别为屏幕大小的 40% 和 30% 的图形窗口，该窗口的颜色为天蓝色（[0,1,1]）的没有默认菜单栏的窗口，其标题为 My New Window，并可以建立图 8-5 所示的菜单系统。

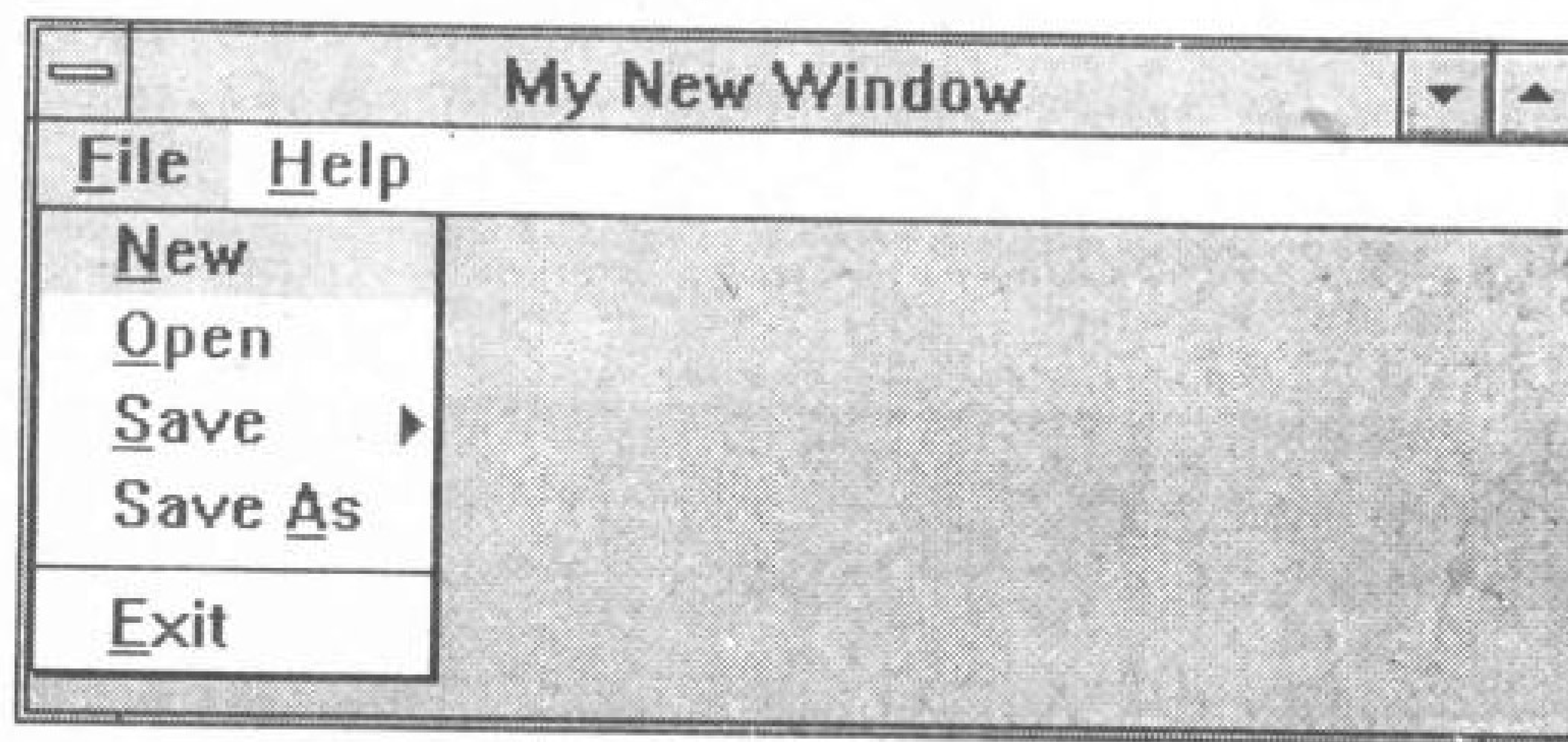


图 8-5　窗口及菜单选项

这时如果选择 File | New 选项，则将在命令窗口上显示 New Item Selected 字样，如果选择 File | Open 这一选项，则将在命令窗口上显示出 Open Item Selected 字样， File | Save 菜单项初始时处于禁选状态，若选择 Help 选项之后则会将此菜单项恢复成可选状态，如果选择 File | Save 这一选项，则将再给出一个新的菜单，其中共有两个子菜单选项 ASCII file 和 Graphics file，如果选择第 1 个选项，则将变量 key 和 k0 分别赋为 3 和 1，然后调用 filesys.m 文件来进行相应的处理（该文件需要另行编写）。如果选择 File | Exit 这一选项，则将关闭当前窗口。如果选择 Help | About... 选项，则在 MATLAB 命令窗口上显示 my help 字样，并将 Save 菜单设置成可选状态。这时还可以使用 Position 选项来测试某一菜单的位置，例如

```
>>  get(msav,'Position')
ans =     3
```

MATLAB 还允许用户动态地添加或删除菜单项，并自由地更改每个菜单项的初始属性，而这些功能可以从其它响应函数中加以定义，下面将通过一个实例来演示这样的

效果。

例 8.3　如果用户想建立一个菜单系统，在调用 File | New 菜单时在 MATLAB 命令窗口中显示 Hello, File New Menu Item Selected 字样，如果按下 File | More Items 菜单项将得到下一级菜单，其中有 Sine Wave 和 Cosine Wave 两个子菜单项，且选择了其中的 Sine Wave 和 Cosine Wave 两个子菜单项，将分别在新打开的图形窗口上显示出正弦和余弦曲线。绘制出余弦曲线之后还将在图形窗口的菜单上添加一个标注为 New Menu Item 的子菜单项。如果按下 File | Quit 菜单则将自动关闭打开的窗口。根据上面的要求用户可以容易地编写出下面的程序段

```
gwin=figure('Color',[0,1,1],'Pos',[0, 0, 300, 200],'Name','My New Window',...
   'NumberTitle','off','MenuBar','none');
hfile=uimenu(gwin, 'Label','&File');
uimenu(hfile, 'Label','&New', 'Call',...
   'disp(''Hello, File New Menu Item Selected'')');
hfilemore=uimenu(hfile, 'Label','More Items');
uimenu(hfilemore, 'Label','Sine Wave', ...
   'Call','t=0:0.1:2*pi; figure; plot(t,sin(t))');
uimenu(hfilemore, 'Label','Cosine Wave', 'Call',['t=0:0.1:2*pi; ',...
   'figure; plot(t,cos(t)); uimenu(hfile, ''Label'',''New Menu Item'')']);
uimenu(hfile, 'Label','&Quit', 'Call','close(gwin)');
```

在这一程序段中，首先打开了一个图形窗口，其菜单系统如图 8-6 所示，若选择其中的每个选项，其效果和前面的要求是一致的，其中这里的回调函数的结构还是比较复杂的，有关回调函数的写法将在后面详细介绍。

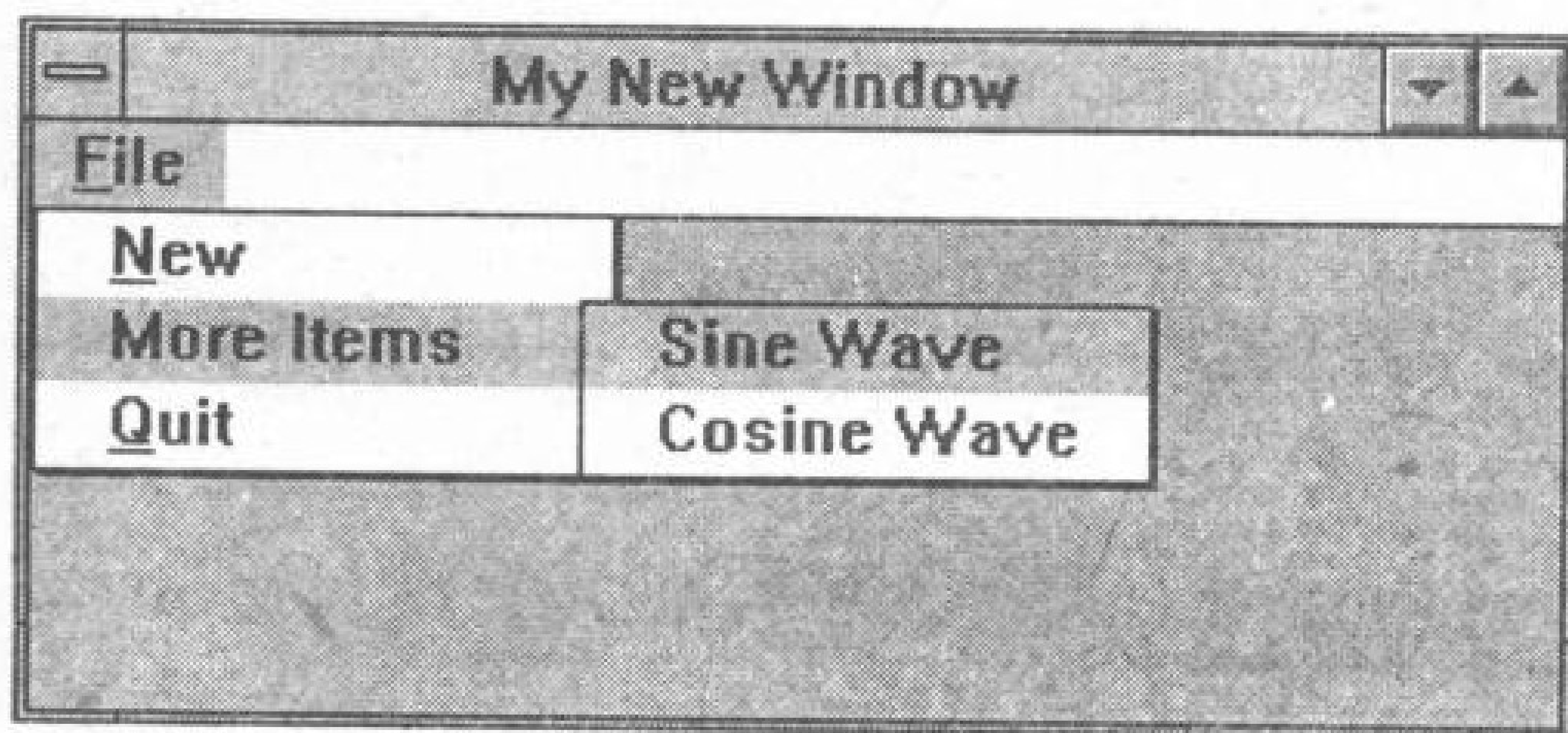


图 8-6　菜单系统实例

由上面的叙述可见，在对窗口菜单进行响应时应该在 Callback 属性的属性值处填写字符串，这个字符串既可以是前面叙述的简单函数名，还可以是复杂的 MATLAB 语句组，这就需要对字符串的写法有所了解，在第 2 章中曾对字符串的一般写法作了简洁的叙述，然而这些叙述往往不能满足用户编写复杂响应的需要，因此这里将进一步介绍字符串书写的附加知识。

- **在字符串中加入单引号**: 字符串是应该由单引号括起来的，所以在字符串中使用单引号时，应该由双重单引号来表示，例如在例 8.3 中回调函数定义中使用了

''Label'',''New Menu Item''

字样来表示单引号，以保证最终的字符串中出现 'Label','New Menu Item' 的字样。

- **构成较长的字符串**: 较长的字符串可以由字符串向量组成 (由 [ ] 括起来)，而在括起来的字符串中还应该用续行符号串接起来，例如在例 8.3 中用到了

```
['t=0:0.1:2*pi; figure; plot(t,cos(t)); ',...
   'uimenu(hfile, ''Label'',''New Menu Item'')'];
```

来表示较长的字符串。其作用是在此菜单项选中时，设置一个时间向量，并绘制出余弦曲线，然后再在 File 菜单上 (句柄为 hfile) 添加一个标注为 New Menu Item 的新菜单项。

8.4 对话框设计方法

8.4.1 对话框的基本元素和实现

对话框也是窗口中最常用的元件，如果用户想和计算机进行交互 (interact), 对话框是最常用的一种手段，因为使用对话框，用户可以通知计算机一些自己的选择，此外还可以将一些参数赋给计算机，而计算机也可以通过对话框将一些信息反馈给用户。在这里将介绍一下对话框及有关的一些基本的元素。

所谓对话框，就是一些用来要求或提供信息的暂时出现的窗口。在对话框上有各种的控制图符和文字，一个典型的对话框如图 8-7 所示。在此对话框中，有自己的窗口边

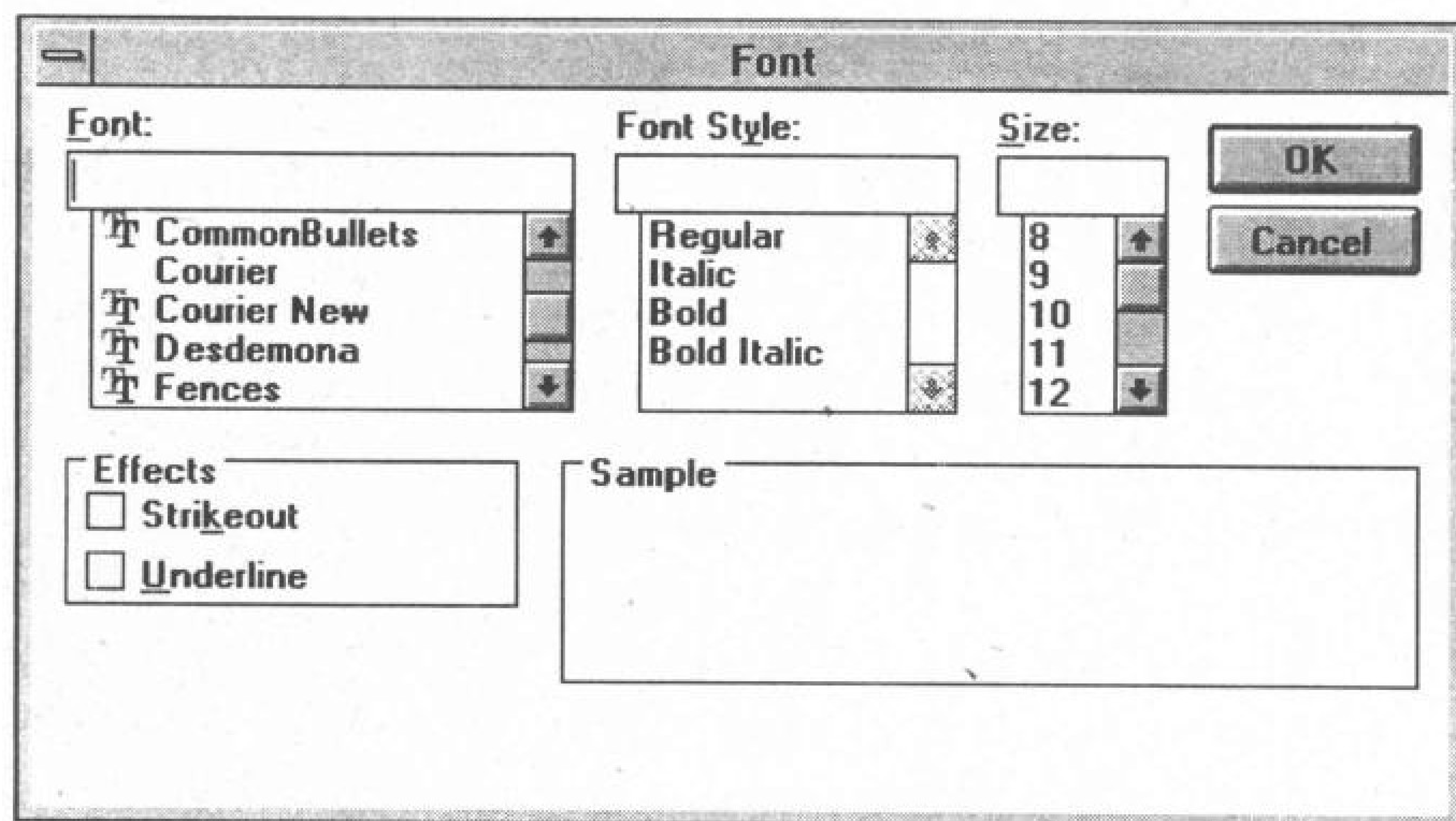


图 8-7 典型对话框实例

界，也可以有其它一些窗口元素，如系统菜单、标题栏和最大最小框，但作为对话框，还应该有一些不同于普通窗口的元件，这就是其控制图符 (graphical controls) 。在图 8-8 中给出了一些常用图符的名称，下面将对其中一些图符作必要的解释。在本书中，将直接使用这些概念：

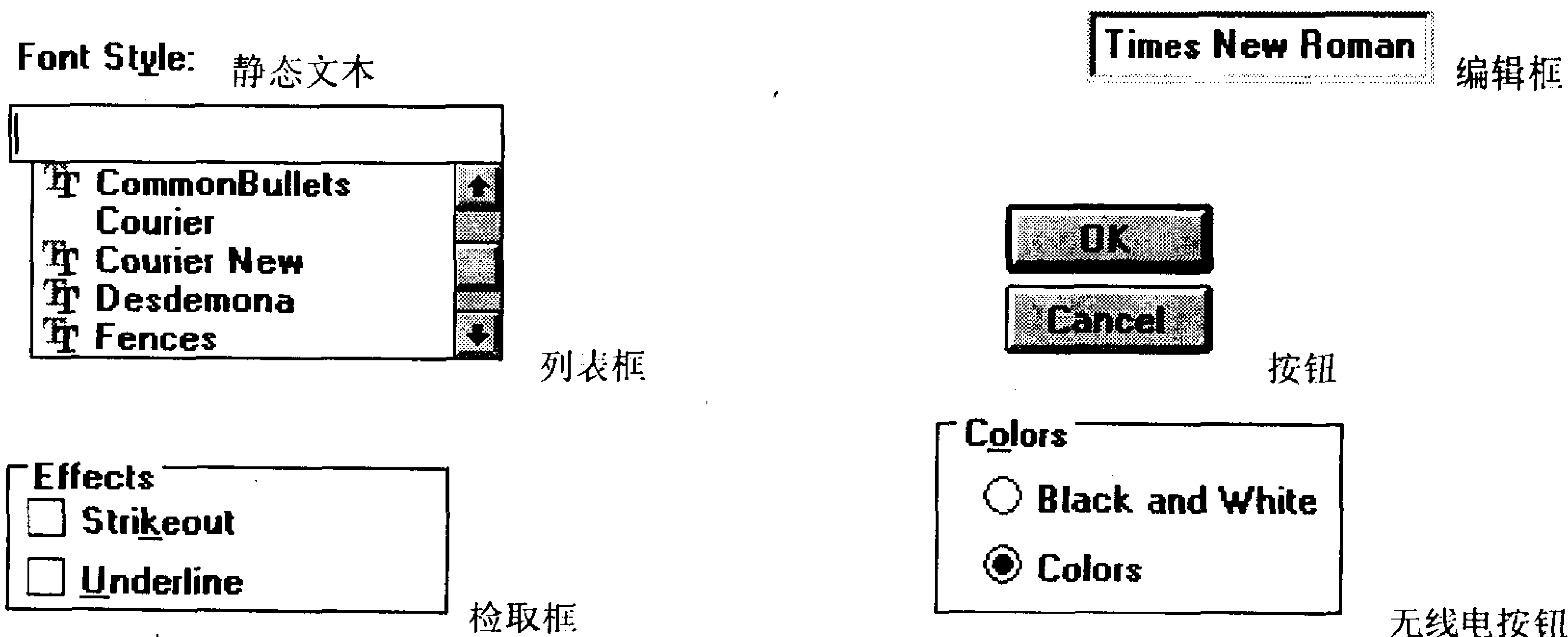


图 8-8　对话框中常用图符

- **静态文本** (static text)：静态文本是用来在对话框窗口之内显示的文字，它一般是用来向用户作信息提示的。例如如果要求用户输入一个数字，可以先用一个静态文本来解释该数字的意义。
- **编辑框** (edit box)：编辑框是一个含有初值的或空白的方框，用户可以在里面填写自己的数据，计算机可以从该框内读取用户提供的信息。
- **列表框** (list box)：列表框列出了可以选择的一些选项，如果选项太多，可以采用垂直滚动条来控制，用户可以方便地从中选择一个选项。在一些特殊情况下，为了不占用过大的空间，则这种列表框可以以一行的形式表示，在右边有一个向下的箭头，如果用户点中此箭头，则将此列表框展开，这种列表框又称为下拉式 (drop down) 列表框。
- **滚动杆** (scroll bar)：滚动杆可以用图示的方式在一个范围内输入一个数量的值。用户可以移动滚动杆中间的游标来改变它对应的参数。
- **按钮** (button)：按钮是对话框中最常用的控制图符，一般说来，一个对话框上至少应该有一个按钮。在按钮上通常有字符来说明其作用，例如 OK 按钮、Yes 按钮和 Help 按钮等。如果用户用鼠标点中一个按钮，则称此按钮为选中状态。
- **无线电按钮** (radio button)：无线电按钮就是一组带有文字提示的选择项，在这一组中通常只能有一个选项被选中，如果用户用鼠标点中了其中一个，则称这一按钮被选中，被选中的按钮在圆的中心有一个实心的黑点，而原来被选中的一个选项就不再处于被选中的状态了，这就象收音机一次只能选中一个台一样，故称作无线电按钮。
- **检取框** (check box)：检取框按钮的作用是和无线电按钮很接近的，它也是一组选择项，所不同的是，检取框一次可以选择多项。

除了这些基本元件之外，在一般图形界面的对话框中还可以带有某种修饰用的元件，如在某一些元件的外面加一个方框等等，这样就使得对话框更富于变化。了解了对话框及控制图符的基本功能，就会更容易地学习后面的内容了。以后再遇到控制图符，则将不再介绍它们的意义或选取方法。



8.4.2 标准对话框的实现与调用

设计出一个高水平的对话框并不是一个简单的事情，为了给使用者提供更大的方便，同时也是为了界面的规范和统一，MATLAB 下给出了若干个标准对话框的直接调用函数，这样就保证用户可以通过尽可能简单的方法对 Microsoft Windows 下的一些通用的对话框进行直接的使用。

- **文件名处理对话框及调用函数**: 如果用户想打开一个已经存在的文件，则在 Microsoft Windows 下最方便的方法是调用一个标准的文件名处理对话框，该对话框可以由 MATLAB 函数 uigetfile() 实现，该函数的调用格式为

```
[文件名, 路径名] = uigetfile(文件类型, 对话框标题, X, Y)
```

其中文件类型为一个字符串，例如如果用户想打开一个 .m 文件，则可以在文件类型处填写 '*.m'，对话框标题也是一个字符串型的变量，用户可以在此处填写任何字符串作为整个对话框标题栏的内容，参数 X，Y 为以像素点为单位的数值，可以由它们方便地给出对话框的初始位置。如果用户使用了如下的命令

```
[myfile, mypath]=uigetfile('*.mat',...
        'This is my test File dialog', 100,100)
```

则将获得一个如图 8-9 所示的对话框，用户可以从这一对话框中用和 Windows 文件对话框操作的同样方法找出一个合适的文件名，然后按下 OK 按钮，这样就会自动

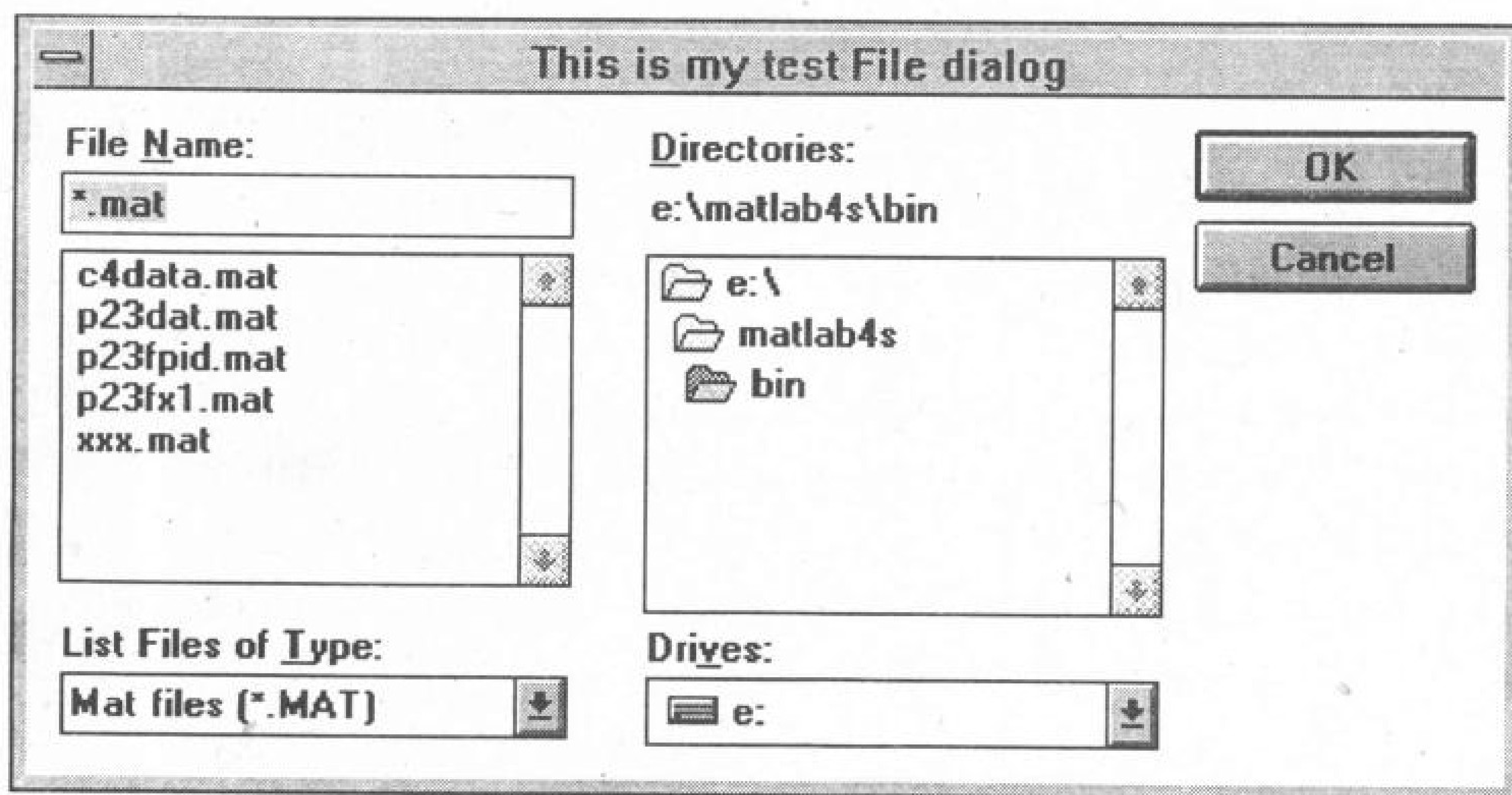


图 8-9 标准文件名处理对话框

返回两个字符串 myfile 和 mypath, 分别给出查找到文件的文件名和路径名，例如如果用户选择了其中的 mydata.mat 文件，则会返回如下的结果

```
myfile =  MYDATA.MAT
```

```
mypath =  E:\MATLAB4S\BIN\
```

亦即调用这一函数之后，就会自动返回两个变量，一个是选中的文件名 MYDATA.MAT，另一个是该文件所在的路径 E:\MATLAB4S\BIN\，并将它们分别赋给所相应的变量，这样 MATLAB 就可以对它们进行进一步处理了。如果用户按下了 Cancel 按钮，则将取消文件名操作的动作。例如如果用户想打开该 MATLAB 数据文件，并想从中将存盘的工作空间变量读入 MATLAB 环境，则可以使用下面的命令：

```
mystr=['load ' mypath, myfile]; eval(mystr)
```

该命令首先构造一个调用文件命令的字符串，然后由 eval() 函数来启动该命令，这样就可以将该文件读入 MATLAB 工作空间了。可见这样的操作是十分简单和方便的。

相应地，如果用户想打开一个文件去输出一些信息，则可以调用 uiputfile() 函数来完成，该函数的调用格式为

[文件名，路径名] = uiputfile(文件类型，对话框标题，X，Y)

可见，这两个函数的调用及功能是极其相似的，用户可以简单地获得一个文件名及路径名，然后对之进行进一步的处理。

- **字体设置对话框及调用函数**: Microsoft Windows 提供了方便的字体设置功能，它在 MATLAB 下也有充分的体现，因为 MATLAB 提供了 uisetfont() 函数，允许用户改变字符及坐标轴字体的形式，该函数的调用格式为

字体句柄 =uisetfont(句柄，对话框标题)

但这样的使用首先要求用户已知要改变内容的句柄。如果不提供句柄变量，则可以由下面的语句进行整体的字体设置。例如若给出下面的命令

```
hFont=uisetfont('My Font Select Dialog')
```

则可以获得如图 8-10 所示的对话框，用户可以从该对话框容易地设置字体、字号及字体风格等相关的信息。字体设置完成之后，将得到一个字体的句柄，用户可以由 get(hFont, 'FontName') 和 get(hFont, 'Size') 等函数的调用格式分别得出选中的字体名称和字号大小，关于字体句柄的其它分量用户可以由 get(hFont) 得出。

- **颜色设置对话框及调用函数**: MATLAB 还提供了 uisetcolor() 函数，通过它可以对对象的颜色进行设置，该函数的调用格式为

颜色值 =uisetcolor(句柄，对话框标题)

这里返回的颜色值是一个 1×3 的向量，这一函数的各个变量和前面介绍的比较类似，这里就不再详细叙述了。如果用户给出一个命令

```
mycol=uisetcolor('My New Color Dialog')
```

则将给出一个如图 8-11 所示的对话框，用户可以从给出的颜色方框中选中一个颜色，



再按下 OK 按钮，这样就可以将该颜色值返回给 mycol 变量，如果用户选择了 Cancel 按钮，则取消颜色设置的动作。

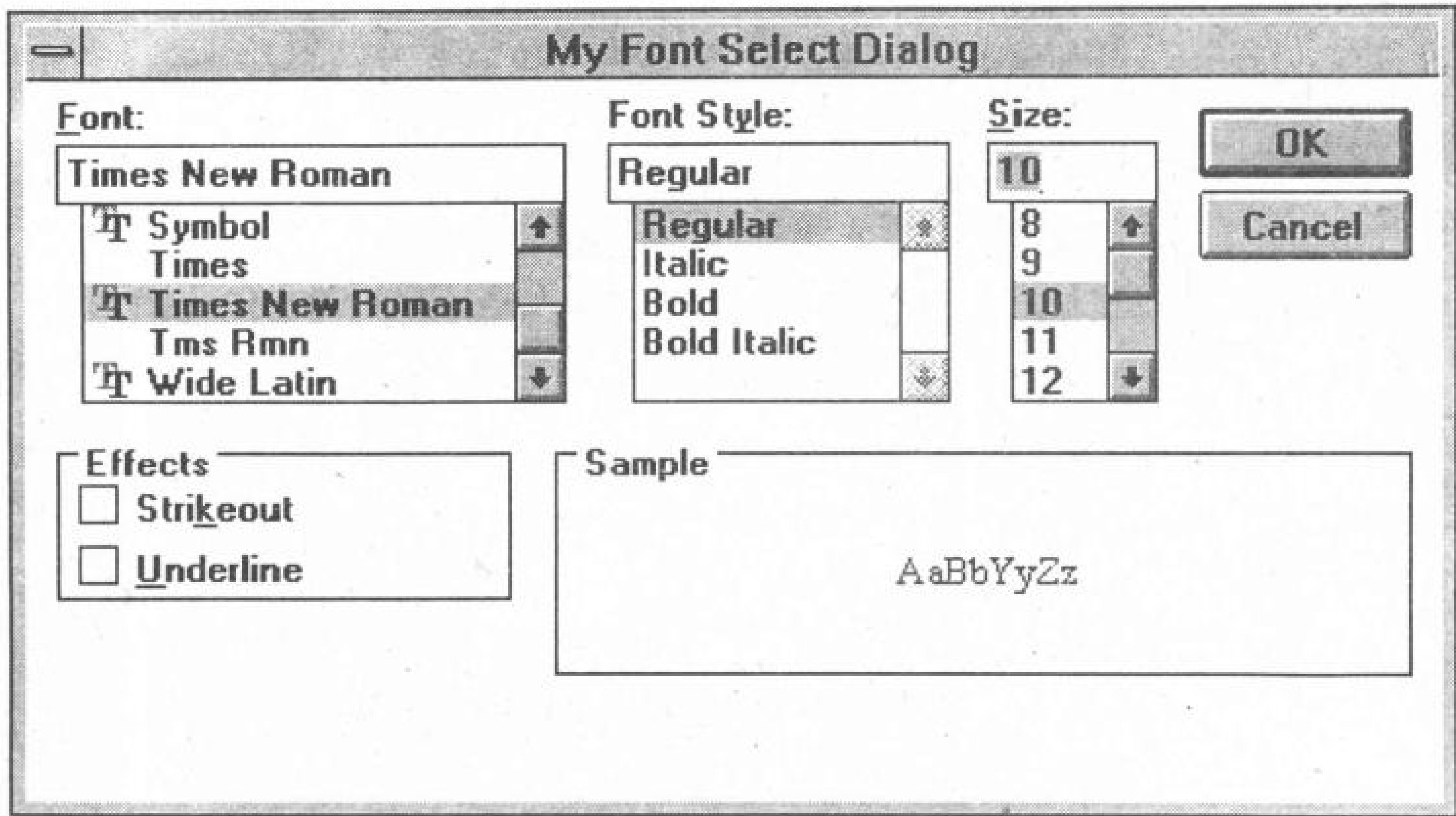


图 8-10 标准字体设置对话框

MATLAB 提供的颜色设置对话框继承了普通 Windows 颜色设置对话框的一些功能，比如如果用户想进一步定义其它颜色，则可以按下 Define Custom Colors (定义用户颜色) 按钮，则将给出一个如图 8-12 所示的对话框，可见该对话框的左半部分和图 8-11 中的对话框是一致的，而右半部分增加了一个调色盘，用户可以在该调色盘上设计出自己所需要的颜色来，然后按下 Add to Custom Colors (添加为用户颜色) 按钮来把选中的颜色填充到左半部分的方格中去，然后重复前面的操作就可以得出选择颜色的句柄了。

8.4.3 一般对话框的设计

除了前面给出的标准对话框以外，用户往往需要设计出具有自己特殊目的的对话框来，这样就需要自己编写对话框调用的程序来完成，MATLAB 提供了方便的对话框元素生成及调用的命令，所用这些命令是由 uicontrol() 函数的调用来处理的，该函数的调用格式为

```
返回句柄 =uicontrol(对话框句柄, 属性 1, 属性值 1, 属性 2, 属性值 2, ···)
```

其中各个属性及可取的值和前面介绍的菜单项属性有些接近，但也不尽相同，这里将着重介绍一些重要的属性，以及和菜单函数不同的属性。

- **Style 属性**: 这一属性用来设置控制元件的风格，它可以决定所设计的控制元件是哪一类元件，亦即是按钮、静态文本，还是其它的元件。MATLAB 4.0 可以支持的各种控制元件风格如表 8-3 所示。

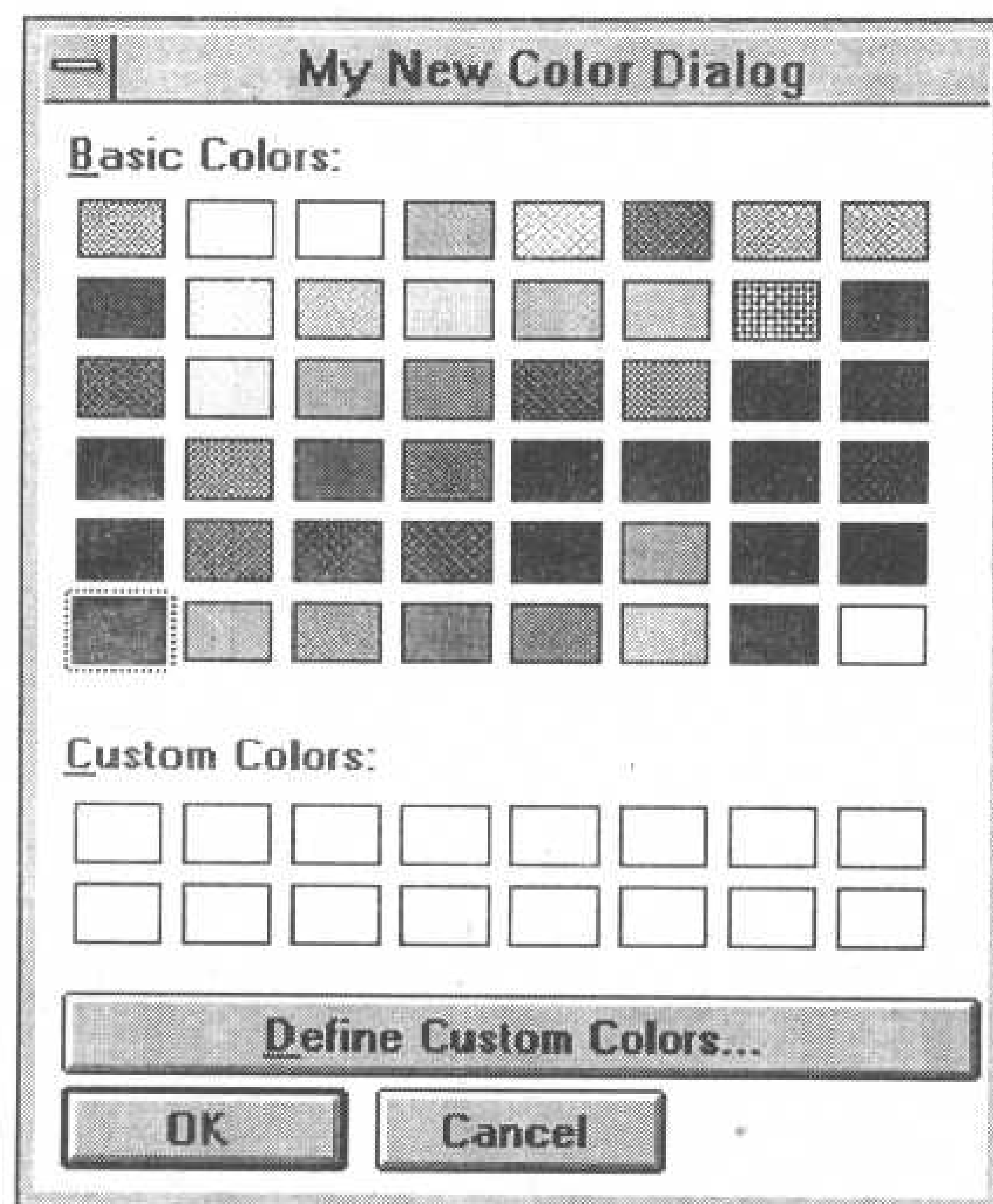


图 8-11　标准颜色设置对话框

- Callback **属性**：和菜单设置函数 uimenu() 函数一样， Callback 属性允许用户建立

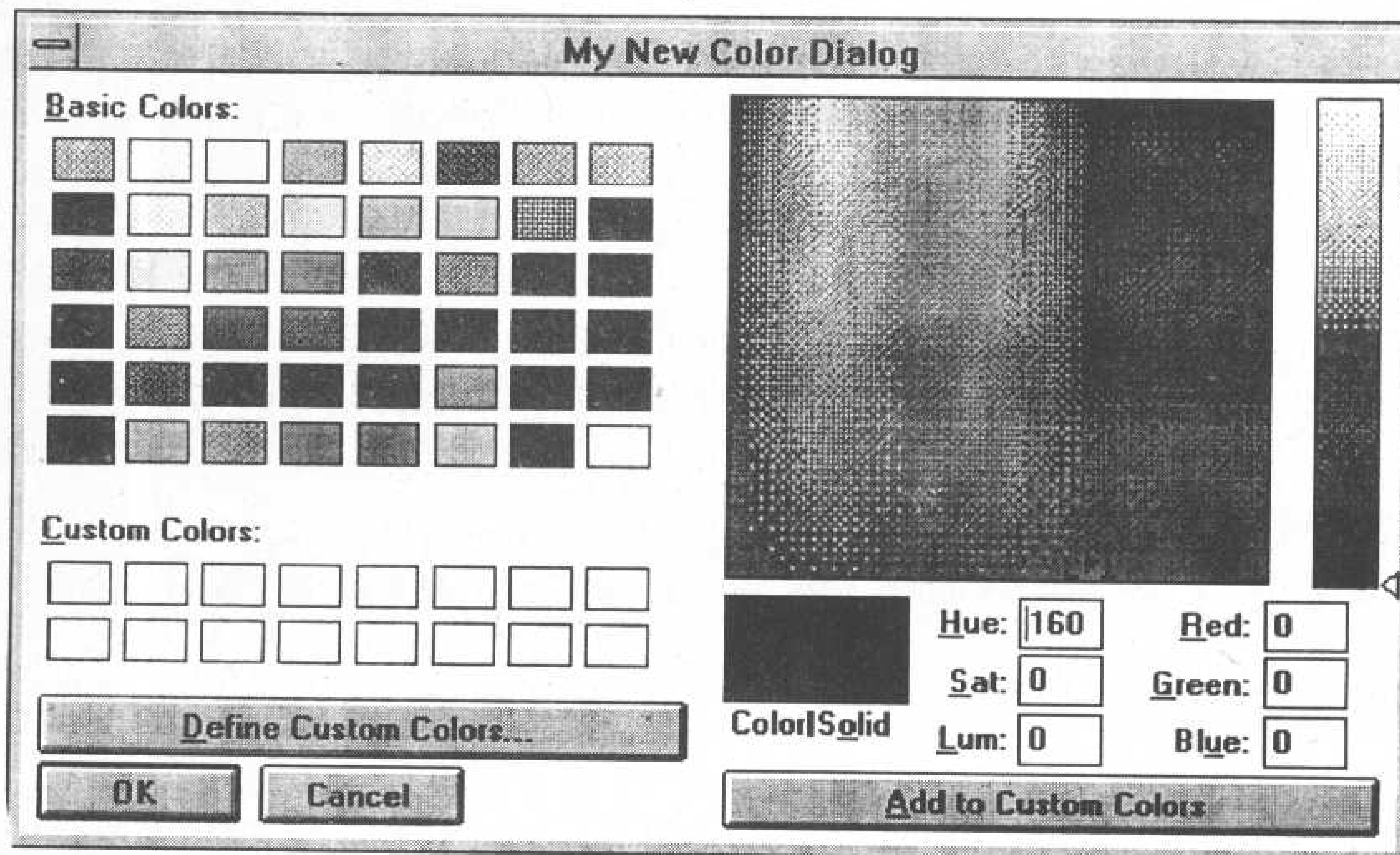


图 8-12　扩充颜色设置对话框

起在对话框元件被选中后的响应命令。

表8-3 对话框控制符风格参数表

关键词	简称	意义
'Pushbutton'	'Push'	按钮
'Radiobutton'	'Radio'	无线电按钮（允许同时选择多项）
'Checkbox'	'Check'	检取框
'Edit'	'Edit'	编辑框
'Text'	'Text'	静态文本
'Slider'	'Slider'	滑标尺
'Frame'	'Frame'	加方框（用于修饰）
'Popupmenu'	'Popup'	类似于标准 Windows 的下拉式列表框

- **String 属性**: 出现在控制元件上的字符串，如在按钮上的说明文字或无线电按钮后面的说明文字等，原则上来说应该可以使用任意的字符串。
- **Position 和 Units 属性**: 它们的选择范围和 set() 函数中定义的是一致的，在这里就不再赘述了。
- **Visible 属性**: 也是用来决定该控制元件初始状态是否可见的，它的属性值可以同样选择为 'on' 和 'off' ，而 'on' 选项是默认的。
- **BackgroundColor 和 ForegroundColor 属性**：分别用来控制该元件的前景和背景颜色的，它们的取值仍然为 3×1 的颜色配比向量。例如对按钮控制元件来说，前景颜色即指按钮上的字符颜色，而背景颜色为整个按钮的颜色。
- **HorizontalAlignment 属性**: 用来决定当前的控制元件在水平方向上的对齐方式，如它的可取属性值为 'left' 、 'center' 和 'right' ，分别表示按左、中和右侧的对齐方式，其中的 'right' 为默认状态。当然这样的选项对单个的控制元件并没有多大的意义，而只对一组元件起作用，比如可以设置一组静态文本说明，而它们位置的 X 起点坐标是一致的，并且它们的宽度是一致的，这样使它们全部选择中间对齐的选项，则可以比较规整地将它们显示出来。

例 8.4 和标准的 Windows 的对话框元素不同，利用 MATLAB 直接建立起来的无线电按钮是可以同时选择多项的，考虑下面给出的程序段

```
gwin=figure('Color',[0,1,1], 'Position',[100,200,400,200],...
   'Name','My Own Program','NumberTitle','off', 'MenuBar','none')
nradios=5;
for i=1:nradios
   radios(i)= uicontrol(gwin, 'Style','Radio','Position',[0.2,0.85-0.15*i, ...
   0.2,0.1], 'Units','normal','String',['''A=''' int2str(i)], 'Back',[1,1,0]);
end
```




如果想在一组无线电按钮中只选择一项，则应该作一些特殊的处理，使它们之间保证一种互斥 (mutual exclusive) 的关系 [2], 例如用户可以在前面的程序段后面填补上下面的程序语句

```
for i=1:nradios
   set(radios(i),'UserData',radios(:,[1:(i-1),(i+1):nradios]));
end
mycall=['me=get(gcf,''CurrentObject''); if (get(me,''Value'')==1),',...
  'set(get(me,''UserData''),''Value'',0), else, set (me,''Value'',1), end'];
set(radios,'CallBack',mycall);
```

在这一程序段中首先对各个无线电按钮添加一个用户数据向量 (UserData 属性值)，用以存放组中其它无线电按钮的句柄，这样在选中一个无线电按钮时会自动地将其它无线电按钮的 Value 属性值设置为 0，以取消相应的无线电按钮选中状态，从而保证总是只有一个无线电按钮被选中，也就是说这些无线电按钮之间有互斥的关系。

例 8.5　用户可以键入下面的程序段

```
A=0.1; gwin=figure('Color',[0,1,1], 'Position',[100,200,400,200],...
   'Name','My Own Program','NumberTitle','off', 'MenuBar','none')
uicontrol(gwin, 'Style', 'Text', 'Position', [0.1, 0.8, 0.8, 0.1],...
   'Units', 'normalized', 'Horizontal', 'center', ...
   'String', 'This is a test dialog box', 'Back', [0,1,1]);
uicontrol(gwin, 'Style', 'Text', 'Position', [0.05, 0.65, 0.9, 0.1],...
   'Units', 'normalized', 'Horizontal', 'center', ...
   'String', 'Designed by Dr Dingyu XUE, 4 June, 1995',...
   'Back', [0,1,1], 'Fore', [1,0,0]);
uicontrol(gwin, 'Style','Frame', 'Position',[0.15,0.15,0.5,0.45],...
   'Units', 'normalized', 'Back',[1,1,0]);
edits=uicontrol(gwin, 'Style','Edit', 'Position',[0.42,0.45,0.2,0.1],...
   'String','0.1', 'Units', 'normalized', 'Back', [0,1,0]);
radios(1)=uicontrol(gwin, 'Style','Radio','Position',[0.2,0.45,0.2,0.1],...
   'Units','normal','String','Enter A', 'Back',[1,1,0]);
radios(2)=uicontrol(gwin, 'Style','Radio','Position',[0.2,0.35,0.2,0.1],...
   'Units','normal','String','A=100', 'Back',[1,1,0]);
radios(3)=uicontrol(gwin, 'Style','Radio','Position',[0.2,0.25,0.2,0.1],...
   'Units','normal','String','A=200', 'Back',[1,1,0]);
S_comm=['if get(radios(2),''Value'')==1, A=100, end; ',...
       'if get(radios(3),''Value'')==1, A=200, end; ',...
       'if get(radios(1),''Value'')==1, ',...
       'A=eval(get(edits,''String'')), end; close(gwin)'];
uicontrol(gwin, 'Style','Push', 'Position',[0.7,0.35,0.2,0.12],...
   'String','Get A', 'Units', 'normalized', 'Call',S_comm);
uicontrol(gwin, 'Style','Push', 'Position',[0.7,0.2,0.2,0.12],...
   'String','Cancel', 'Units', 'normalized', 'Call','close(gwin)');
```

这样就可以建立起如图 8-13 所示的对话框，然后在该对话框上分别建立静态文本显示 This is a test dialog box 和 Designed by Dr Dingyu XUE, 4 June, 1995，并在该对话框中绘制一个修饰用方框，在

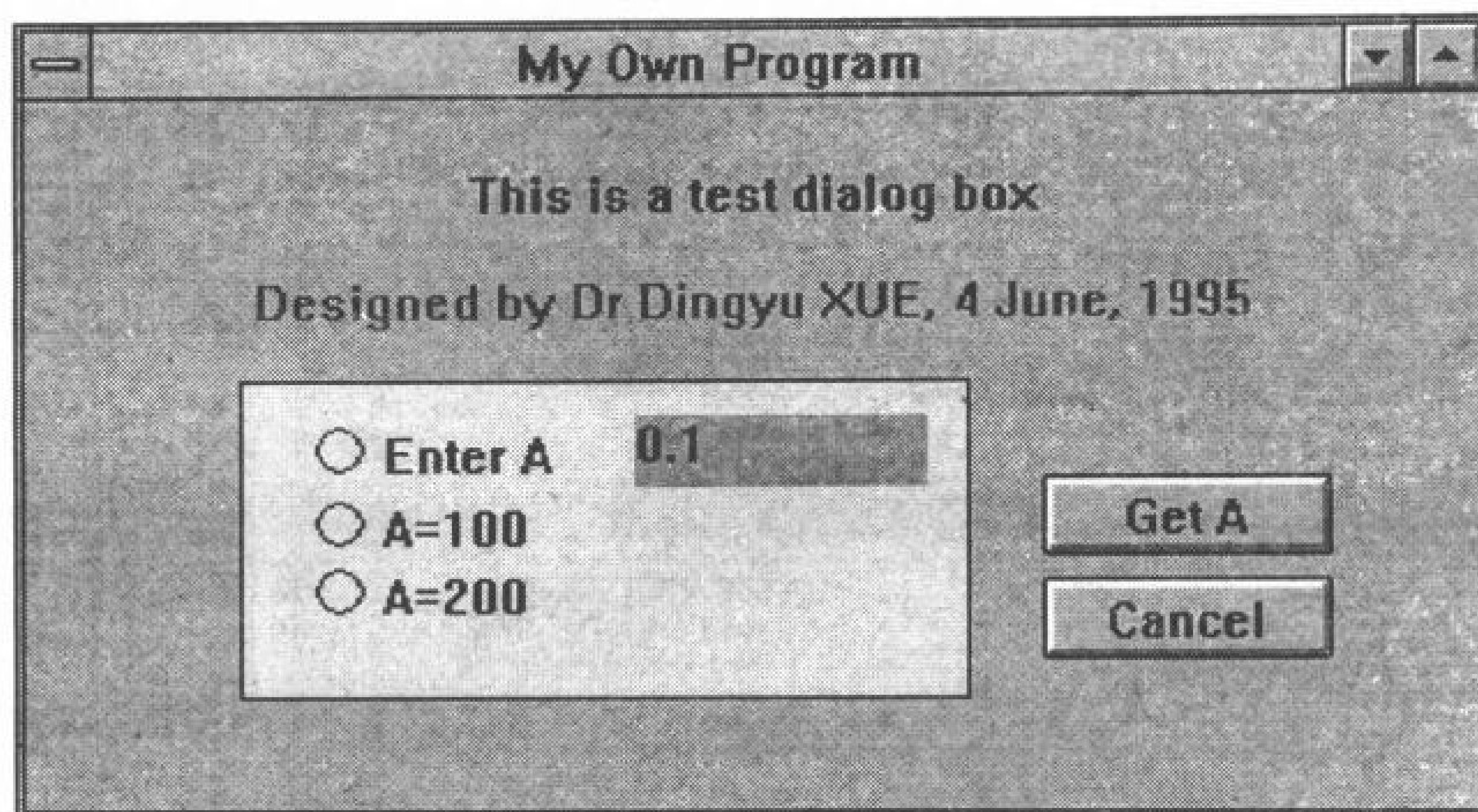


图8-13 对话框的一个应用实例

该方框内还加入了 3 个无线电按钮 (其句柄分别为 radios(1) → radios(3)) 与一个编辑框 (句柄为 edits), 最后在该对话框中加入了两个按钮, Cancel 和 Get A, 若选择了 Cancel 按钮, 则将关闭此对话框, 并不对 A 变量的参数进行任何的修改, 若按下了 Get A 按钮, 则将执行 S_comm 回调命令, 这一组回调命令中首先判断是哪一个无线电按钮被选中 (被选中的无线电按钮其 Value 属性值为 1, 否则为 0), 并对变量 A 进行相应的赋值。注意, 这里的各个无线电按钮并没有互斥的关系, 所以同时可以选中几个, 这样 A 变量的最终取值将取决于 S_comm 中语句的顺序, 当然结合前面介绍的互斥无线电按钮的设置, 可以对上面的程序稍作改动, 即将下面的语句

```
nradios=3;
for i=1:nradios
   set(radios(i),'UserData',radios(:,[1:(i-1),(i+1):nradios]));
end
mycall=['me=get(gcf,''CurrentObject''); if (get(me,''Value'')==1),',...
  'set(get(me,''UserData''),''Value'',0), else, set (me,''Value'',1), end'];
set(radios,'CallBack',mycall);
```

插入到上面程序段中相应的部分, 就可以保证这些无线电按钮之间有互斥的关系, 这时 A 变量的取值将与 S_comm 中回调语句的顺序无关。

例 8.6 假设想建立如图 8-14(a) 所示的程序界面, 其中 File 菜单的内容如图 8-14(b) 所示, 它负责模型的读入与存盘, 完成指定程序功能的程序如下所示

```
color0=[1,1,1]; color1=[1,0,1];
gwin=figure('Color',color0, 'Position',[100,200,400,200],...
   'Name','My Own Program','NumberTitle','off', 'MenuBar','none')
mfile=uimenu(gwin,'Label','&File');
ss1=['[ff,pp]=uigetfile(''*.mat'',''Get Model''); eval([''load '' pp ff]); ',...
   'set(uinum,''String'',mat2str(num)); set(uiden,''String'',mat2str(den));'];
ss2=['[ff,pp]=uiputfile(''*.mat'',''Save Model''); eval([''save '' pp ff ',...
   ''' num den''])'];
uimenu(mfile, 'Label','&Open Model', 'Call',ss1);
```

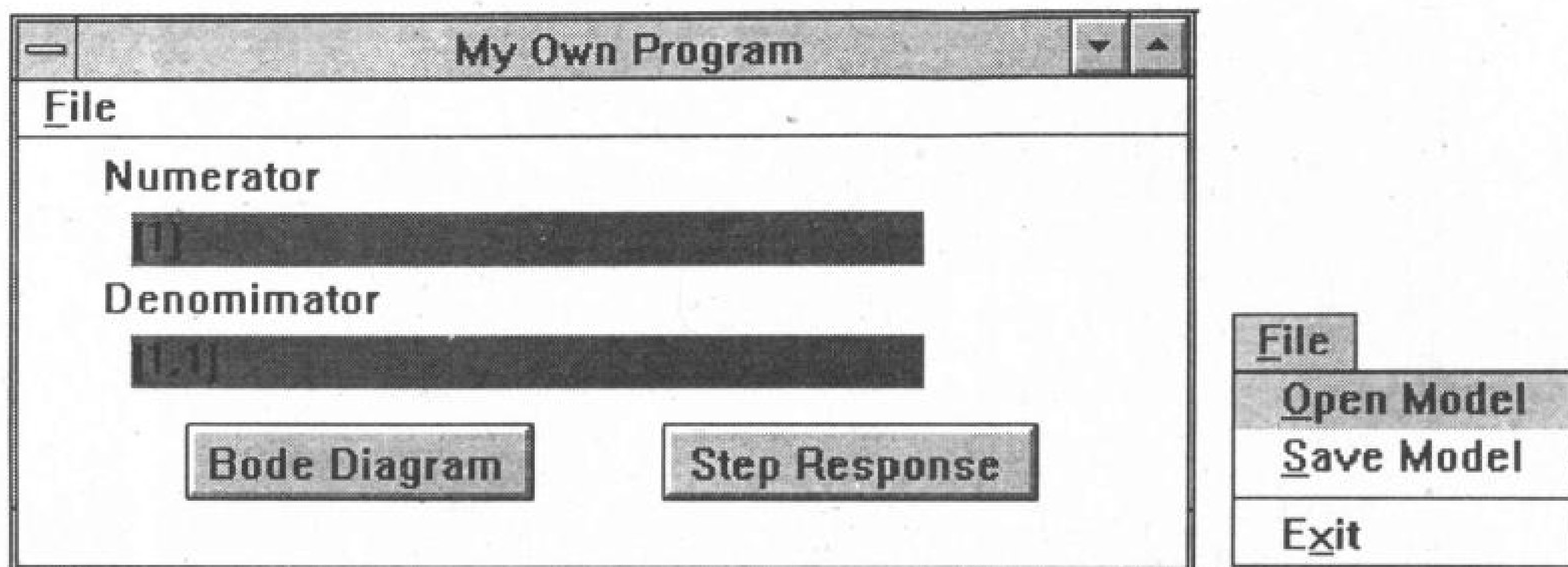


图8-14 界面设计应用实例及菜单

```
uimenu(mfile, 'Label','&Save Model', 'Call',ss2);
uimenu(mfile, 'Label','E&xit', 'Separator','on', 'Call','close(gwin)');
uicontrol(gwin,'Style','Text','Pos',[10,170,90,18],'String','Numerator',...
   'Back',color0);
uicontrol(gwin,'Style','Text','Pos',[10,130,110,18],'String','Denomimator',...
   'Back',color0);
uinum=uicontrol(gwin,'Style','Edit','Pos',[30,150,250,18],'String','[1]',...
   'Back',color1);
uiden=uicontrol(gwin,'Style','Edit','Pos',[30,110,250,18],'String','[1,1]',...
   'Back',color1);
ss0='vn=get(uinum,''String''); vd=get(uiden,''String'');';
ss01='eval(ss0); num=eval(vn); den=eval(vd); figure;';
ss3=[ss01 'bode(num,den);']; ss4=[ss01 'step(num,den);'];
uicontrol(gwin,'Style','Push','Pos',[50,50,110,25],'String',...
   'Bode Diagram','Call',ss3);
uicontrol(gwin,'Style','Push','Pos',[200,50,120,25],'String',...
   'Step Response','Call',ss4);
```

这一程序中 File 菜单是基于标准文件名处理对话框调用的，如果用户选择 Open Model 菜单项，则会打开标准文件输入对话框读入一个文件名，然后会自动地将系统的传递函数 num 和 den 提供 mat2str() 函数读入转换成相应的字符串，其中 mat2str() 函数是作者编写的，其内容如下

```
function str=mat2str(vv)
[nr,nc]=size(vv); str=[];
for i=1:nr
   str=[str num2str(vv(i,1))];
   for j=2:nc, str=[str ',' num2str(vv(i,j))]; end; str=[str ';'];
end
str=['' '[' str(1:length(str)-1) ']' ];
```

其作用是将一个已知的矩阵用 MATLAB 标准的字符串格式表示出来，用户可以通过这里的程序将传递函数模型对应的字符串填写到相应的编辑框中去，传递函数模型的参数还可以通过编辑框中

数据的修改而直接完成，这时若用户按下 Bode Diagram 按钮，则会将系统的 Bode 图自动地绘制出来，若按下 Step Response 按钮，则会将系统的阶跃响应曲线自动地绘制出来，可见由这样一个简单的界面和有限几条 MATLAB 语句就可以完成较强的系统分析功能。

8.5 应用实例 – Control Kit

由于 MATLAB 这样的专用语言使用是十分方便的，所以作者曾用此语言编写了一个名为 Control Kit 的反馈控制理论教学软件 [3]，这一软件作为英国 Rapid Data Ltd 软件公司的商品在英国和西欧一些大学和研究机构中有成功的应用。当时的 Control Kit 版本是基于 PC-MATLAB 来开发的，所以并没有涉及到图形界面的使用。在 MATLAB 4.0 推出以后，作者和 Rapid Data Ltd 公司的 Ole Sorensen [1] 分别独立地对这一软件进行了改写，将它设计成用户友好的 Windows 版本。本章的附录中将给出作者改写的 Control Kit 中部分程序清单，并介绍和图形用户界面有关的程序设计问题，以资参考。这里列出的程序在功能上也是绘制系统的 Bode 图和阶跃响应曲线的，所以在程序功能上和例 8.6 中所示的程序是接近的。

首先进入 MATLAB 环境，然后在提示符下键入 ctrlkit，这时将自动执行反馈控制系统教学软件 Control Kit，并得出如图 8-15 所示的程序界面，用户可以分别输入各个模块的

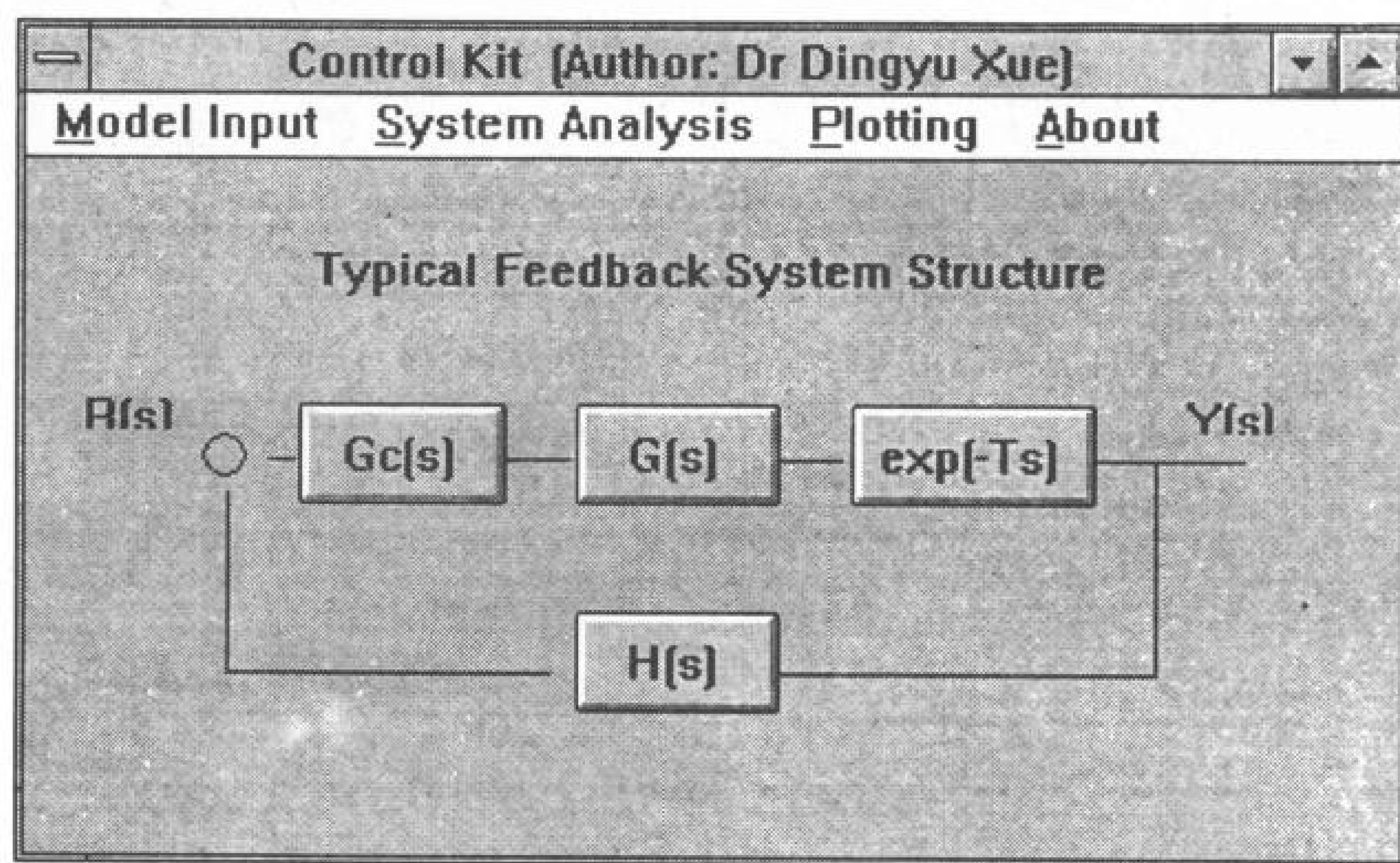


图 8-15 Control Kit 程序界面

数学模型，例如若用户想输入受控对象 $G(s)$ 的数学模型，则可以用鼠标器左键点取图中的 $G(s)$ 字样，这时将得出一个如图 8-16 所示的对话框，提示用户输入系统的传递函数模型 1)，用户可以通过此对话框按照 MATLAB 的格式给出系统传递函数的分子和分母系数向量，例如图 8-16 中表示的数学模型为 $G(s)=(s^3+7s^2+24s+24)/(s^4+10s^3+35s^2+50s+24)$，

1) Control Kit 提供了各种各样的数学模型输入方法，包括传递函数输入、状态方程输入、结构图输入以及 SIMULINK 输入等，所以从理论上说，它可以用来处理各种复杂程度的数学模型。在本书中的演示例子中由于篇幅所限，只能处理简单的 SISO 传递函数模型，其它的输入方法都是空的框架，其它的很多部分也因考虑篇幅进行了简化。

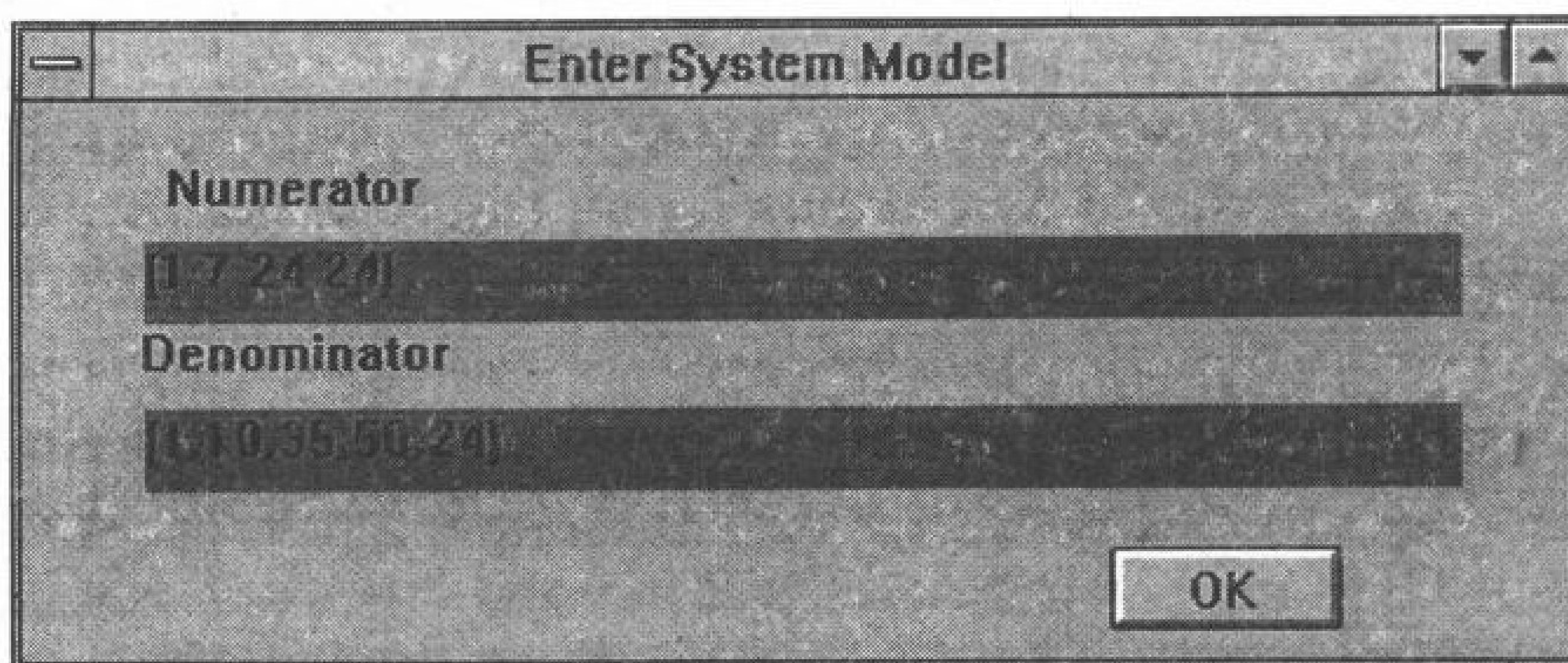


图 8-16 传递函数模型输入对话框

填写了模型之后，按下 OK 按钮，这时程序会自动地将用户输入的模型读入计算机，具体地说，受控对象模型将赋给 G_num，G_den, 控制器模型将赋给 Gc_num，Gc_den, 而反馈模型将赋给 H_num，H_den ，赋值完成后，将关闭图 8-16 所示的对话框。

用户可以对此系统进行计算机辅助分析，例如打开 System Analysis (系统分析) 菜单，见图 8-17(a), 这时用户可以从中选择各种分析方式 (限于篇幅，故其中一些分析功能

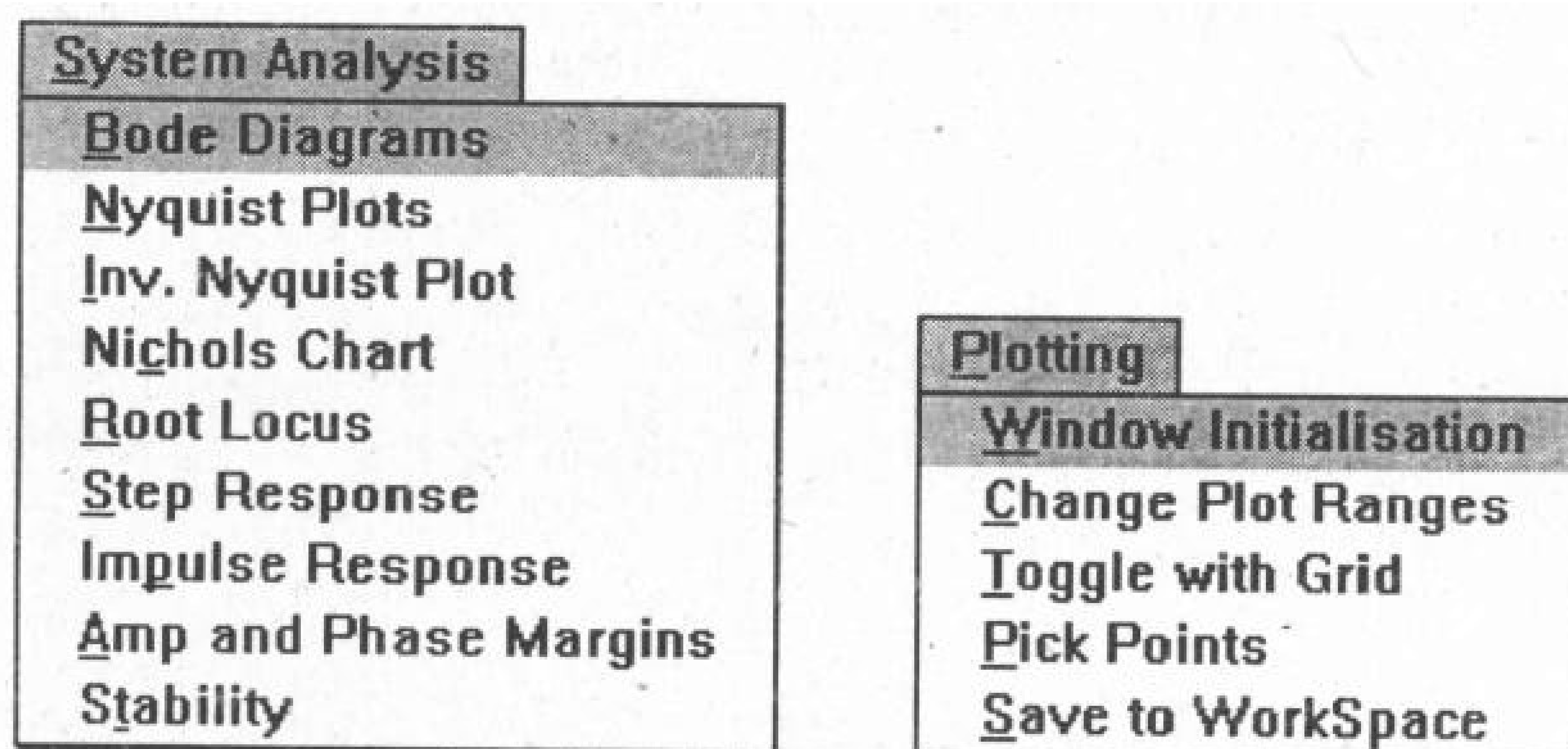


图 8-17 Control Kit 的两个实用菜单

只给出空的框架)，就可以得出系统的响应曲线。例如若选择 Step Responses，则将绘制出阶跃响应曲线，若选择了 Bode Diagram 则将绘制出系统的 Bode 图。系统图形的进一步修饰可以由图 8-17(b) 中所示的 Plotting 菜单中的选项来完成，例如若想获得带有网格线的坐标图形，则可以选择其中的 Toggle with Grid, 若想用鼠标选择并显示某一坐标处的参数则可以选择 Pick Point 选项来实现。

附录 8.A Control Kit 部分程序清单

由于篇幅所限，不可能将整个 Control Kit 软件的清单列出来，这里作者对该程序作了一些压缩，将不是十分必要的一些功能用空的框架来表示，并略去了一些常用的功能，从而使得程序并不是十分庞大。下

面将列出简化版本中的程序清单，以资参考。首先，Control Kit 软件的主程序名为 ctrlkit.m, 其清单如下

```
global WinWidth WinHeight gerr ginfo gmain
col0=[1 0 0]; col1=[0 1 1]; screen=get(0,'ScreenSize');
WinWidth=screen(3); WinHeight=screen(4);
gmain=figure('Color',col1,'pos',[5 .5*WinHeight .5*WinWidth .4*WinHeight], 'Name',...
   'Control Kit  (Author: Dr Dingyu Xue)','NumberTitle','off','MenuBar','none');
newmodel;
shwmenus;
shwstruc;
```

这里首先获得显示系统的分辨率，然后打开一个主窗口 (句柄为 gmain), 再调用相关的一些下一级程序，其中调用的 newmodel.m 程序允许用户对整个系统作初始化，它的清单如下

```
col=[1;0;1];
ginfo=figure('Color',col,'pos',[0.4*WinWidth 0.02*WinHeight,0.59*WinWidth,...
   0.25*WinHeight],'Name','Extra Information','NumberTitle','off','MenuBar','none');
Cmodel=0; Gmodel=0; Hmodel=0; t_delay=0; G_num=[]; G_den=[]; Gnum_tmp='[1]'; Gden_tmp='[1 1]';
Gc_num=[]; Gc_den=[]; Gcnum_tmp='[1]'; Gcden_tmp='[1]'; H_num=[]; H_den=[]; Hnum_tmp='[1]';
Hden_tmp='[1]'; w_range=logspace(-1,1);  t_range=0:.1:10; k_range=0:.1:5; SysModel=0; ks=1;
me=[]; dly_tmp='0'; WinGx1=0.4*WinWidth; WinGy1=0.38*WinHeight; WinGw=0.6*WinWidth;
WinGh=0.6*WinHeight; igrid=0; ipp=0; kredraw=0;
kstep=0; kbode=0; knyqs=0; knichl=0; krtlc=0; kinvn=0; kimpu=0;
```

该程序段打开一个信息窗口 (句柄为 ginfo), 然后对将要用到的各个变量进行初始赋值。主程序 ctrlkit.m 中还有两个其它的程序，其中 shwmenus.m 用于建立整个程序的菜单系统，其清单为

```
mmain=uimenu(gmain,'label','&Model Input');
   uimenu(mmain,'label','New','callback','newmodel');
   mmodl=uimenu(mmain,'label','Enter/Modify');
      uimenu(mmodl,'label','&Transfer Function','callback','key=1; k1=1; promodel')
      uimenu(mmodl,'label','&State Space','callback','key=1; k1=2; promodel')
      uimenu(mmodl,'label','Pole-Zero Model','callback','key=1; k1=3; promodel')
      uimenu(mmodl,'label','Block Diagram','callback','key=1; k1=4; promodel');
      uimenu(mmodl,'label','SIMULINK Model','callback','key=1; k1=5; promodel')
      uimenu(mmodl,'label','Model Validation','callback','key=1; k1=6; promodel')
   uimenu(mmain,'label','Open','callback','key=2; promodel')
   uimenu(mmain,'label','Save','callback','key=3; promodel')
   uimenu(mmain,'label','Save As','callback','key=4; promodel')
   uimenu(mmain,'label','Show Model','callback','key=5; promodel')
manalys=uimenu(gmain,'label','&System Analysis');
   uimenu(manalys,'label','Bode Diagrams', 'callback','iplot=1; analys');
   uimenu(manalys,'label','Nyquist Plots','callback','iplot=2; analys');
   uimenu(manalys,'label','Inverse Nyquist Plot','callback','iplot=3; analys')
   uimenu(manalys,'label','Nichols Chart','callback','iplot=4; analys')
   uimenu(manalys,'label','Root Locus','callback','iplot=5; analys')
   uimenu(manalys,'label','Step Response','callback','iplot=6; analys')
   uimenu(manalys,'label','Impulse Response','callback','iplot=7; analys')
   uimenu(manalys,'label','Amp and Phase Margins','callback','iplot=8; analys')
```

```
    uimenu(manalys,'label','Routh Table and Stability','callback','iplot=9; analys')
    uimenu(manalys,'label','Analytical Solutions','callback','iplot=10; analys')
  mplot=uimenu(gmain,'label','&Plotting');
    uimenu(mplot,'label','Window Initialisation','callback','key=1; pltcom;')
    uimenu(mplot,'label','Change Plot Ranges','callback','key=2; pltcom;')
    uimenu(mplot,'label','Toggle with Grid','callback','key=3; pltcom;')
    uimenu(mplot,'label','Pick Points','callback','key=4; pltcom;')
    uimenu(mplot,'label','Save to WorkSpace','callback','key=5; pltcom;')
  uimenu(gmain,'label','&About','callback','ckabout');
```

可见这里的程序结构十分明显，每个菜单项对应的回调函数关系也是很易弄清的，所以在这里无需太多的解释。总之，若选择了 Model Input 菜单，则会自动调用 promodel.m 程序来对系统模型作某种处理，而选择了 System Analysis 则会调用 analys.m 程序来求取系统的响应，若选择了 Plotting 菜单则会自动调用 pltcom.m 程序来进行绘图处理，若选择了 About 菜单项则会调用 ckabout.m 程序给出 Control Kit 程序的有关说明，该程序的清单如下：

```
  color=[0;1;0]; gabout=figure('Color',[0;1;1],'Name','Control Kit (MS-Windows version)',...
     'NumberTitle','off','MenuBar','none','Resize','off',...
     'pos',[0.3*WinWidth 0.2*WinHeight 0.4*WinWidth 0.3*WinHeight])
  uicontrol('style','frame','unit','normal',...
      'pos',[.05 .05 .9 .9],'BackgroundColor',color)
  pos=[0.1,0.75,0.8,0.1]; shwtext(pos,color,'Control Kit for Feedback Control Course');
  pos(2)=0.6; shwtext(pos,color,'ver 2.1, 1989-1994');
  pos(2)=0.45; shwtext(pos,color,'Author: Dr Dingyu Xue')
  pos(2)=0.3; shwtext(pos,color,'Northeastern University, PRC')
  uicontrol('style','push','unit','normal','pos',[.45 .11 .18 .14],...
     'string','OK','call','close(gabout)')
```

这里调用了 shwtext.m 函数来在指定的图形窗口上显示文字信息，该函数的清单如下

```
  function shwtext(pos,color,textnm)
  uicontrol('Style','text','Position',pos,...
    'Back',color,'Units','normalized','String',textnm);
```

可见该函数在调用时需要用户提供 3 个数据，第 1 个数据 pos 为要显示数据的位置向量 (单位是 0-1 间的小数)，第 2 个数据 color 为要显示数据的背景颜色向量，而第 3 个参数 textnm 为要显示的字符串。

再回顾主程序段中调用的 shwstruc.m 程序，该程序用于在主窗口上绘制出反馈控制系统的结构图，并在按下每一个模块时作出相应的响应，该程序的清单如下

```
figure(gmain)
uicontrol('Style','text','pos',[0.1 0.78 0.8 0.1],'Units','normalized',...
   'String','Typical Feedback System Structure','Back',col0,'Call','ks=1;')
uicontrol('style','frame','unit','normal','pos',[.03 .18 .91 .6],'Back',col1)
uicontrol('Style','Push','pos',[0.2 0.5 0.15 0.15],'Units','normal','String',...
   'Gc(s)','Call','ks=2; if(Cmodel==1), key=5; else, key=1; k1=1; end; promodel')
uicontrol('Style','Push','pos',[0.4 0.5 0.15 0.15],'Units','normalized','String',...
   'G(s)','Call','ks=1; if(Gmodel==1), key=5; else, key=1; k1=1; end; promodel')
uicontrol('Style','Push','pos',[0.6 0.5 0.18 0.15],'Units','normal','String',...
   'exp(-Ts)','Call','errdisp(''Not in Demo'')')
uicontrol('Style','Push','pos',[0.4 0.2 0.15 0.15],'Units','normalized','String',...
```




```
    'H(s)','Call','ks=3; if(Hmodel==1), key=5; else, key=1; k1=1; end; promodel')
uicontrol('Style','Push','pos',[0.35 0.57 0.05 0.005],'Units','normal')
uicontrol('Style','Push','pos',[0.55 0.57 0.05 0.005],'Units','normal')
uicontrol('Style','radio','pos',[0.13 0.52 0.05 0.1],'Units','normal','Back',col1)
uicontrol('Style','push','pos',[0.15 0.27 0.25 0.005],'Units','normal')
uicontrol('Style','push','pos',[0.15 0.26 0.005 0.28],'Units','normal')
uicontrol('Style','push','pos',[0.82 0.27 0.005 0.3],'Units','normal')
uicontrol('Style','push','pos',[0.05 0.57 0.08 0.005],'Units','normal')
uicontrol('Style','push','pos',[0.16 0.57 0.04 0.005],'Units','normal')
uicontrol('Style','push','pos',[0.55 0.27 0.27 0.005],'Units','normal')
uicontrol('Style','Push','pos',[0.78 0.57 0.1 0.005],'Units','normal')
uicontrol('Style','text','pos',[.04 0.61 0.08 0.06],'Units','normal','string','R(s)','Back',col1)
uicontrol('Style','text','pos',[.84 0.61 0.08 0.06],'Units','normal','string','Y(s)','Back',col1)
```

该程序绘制出了系统的结构图，在这里所用的单位制为归一值 (normal), 程序段中还调用了 errdisp.m 函数来显示错误信息，该函数的清单为

```
function errdisp(nametx)
global gerr WinWidth WinHeight;
gerr=figure('Color',[1;0;0],'pos',[0.25*WinWidth,0.17*WinHeight, 0.375*WinWidth,...
   0.15*WinHeight],'Name','Error Message','NumberTitle','off','MenuBar','none');
shwtext([0.01 0.5 0.98 0.3],[1;0;0],nametx);
uicontrol('Style','Push','pos',[0.4 0.2 0.18 0.3],...
   'Units','normal','String','OK','Call','close(gerr)')
```

在前面的程序段中我们注意到，若想输入系统的模型往往要调用 promodel.m 程序来完成，在此演示软件中 promodel.m 的内容为

```
if key==1
   kx=0; key0=k1;
   if (k1<=3)
      if (ks==1), num_tmp=Gnum_tmp; den_tmp=Gden_tmp; getmodel;
      elseif (ks==2), num_tmp=Gcnum_tmp; den_tmp=Gcden_tmp; getmodel;
      elseif (ks==3), num_tmp=Hnum_tmp; den_tmp=Hden_tmp; getmodel; end
   else, errdisp('Not in the Demo'); end
elseif key==2,
   [FileName,PathName]=uigetfile('*.mat','Enter Open File Name');
   if (FileName~=0)
      FileName=[PathName  FileName];  newmodel; eval(['load ' FileName]); SysModel=1;
   else, errdisp('No File Name Provided! Model not Loaded'); end
elseif (key==3 | key==4)
   if SysModel==0, errdisp('No System Model!');
   elseif (key==4 | FileName==[]), kk=1;
      [FileName,PathName]=uiputfile('*.mat','Save File Name');
      if (FileName~=0), FileName=[PathName FileName];
         eval(['save ' FileName ' t_range k_range w_range SysModel Gmodel Cmodel Hmodel ',...
           't_delay G_num G_den H_num H_den Gc_num Gc_den Gcnum_tmp Gcden_tmp Gnum_tmp ',...
           'Gden_tmp Hnum_tmp Hden_tmp igrid dly_tmp']);
```

```
        else, errdisp('No File Name Provided: Model Not Saved!'); end
    end
else, errdisp('Not in Demo'); end
```

这里调用了 getmodel.m 函数来读取系统的数学模型，该程序段如下

```
Smodel=1; gmodl=figure('Color',[0;1;1],'pos',[0.25*WinWidth 0.33*WinHeight,...
    0.53*WinWidth 0.4*WinHeight],'Name','Enter System Model',...
    'NumberTitle','off','MenuBar','none','Clipping','off');
if key0==1,
   pos=[0.05 0.85 0.25 0.1]; namtext='Numerator';
   pos2=[.08 .74 .85 .1]; k=2; vec0=num_tmp; getvec;
   pos=[0.05 0.6 0.25 0.1]; namtext='Denominator';
   pos2=[.08 .49 .85 .1]; k=3; vec0=den_tmp; getvec;
   uicontrol('Style','Push','Position',[0.7 0.05 0.15 0.13],'Units','normalized','String',...
      'OK','Callback', ['num0=val2; den0=val3; close(gmodl); if (ks==1) G_num=num0;'...
      'G_den=den0; Gmodel=1; elseif (ks==2) Gc_num=num0; Gc_den=den0; Cmodel=1;',...
      'elseif (ks==3) H_num=num0; H_den=den0; Hmodel=1; end; SysModel=1;']);
else, errdisp('Not in Demo');  end
```

由于篇幅所限，故这里只介绍传递函数模型的读入方法，尽管 Control Kit 提供了丰富的控制系统模型读入方法，并可以处理任意复杂的模型。注意在 getmodel.m 程序中还调用了 getvec.m 程序来获得一个向量，该程序的清单为

```
eval(['val' int2str(k) '=eval(vec0);'])
shwtext(pos,[0;1;1],namtext);
aa0=['uicontrol(''Style'',''edit'',''Position'',pos2,',...
   '''Units'',''normalized'',''String'',vec0,''call'',''val'];
eval(['a' int2str(k) '=' aa0 int2str(k) '=eval(get(a',int2str(k) ',''''String''''));'');'])
```

当用户选择了 System Analysis 菜单项，则会自动地给出系统分析的选项，包括 Bode 图，Nyquist 图等频率响应，也包括阶跃响应、脉冲响应等时域响应，在这一简化的版本中只允许绘制 Bode 图形和阶跃响应曲线，这时将调用 analys.m 程序，该程序的清单如下

```
if (iplot==1 | iplot==6), num=G_num; den=G_den;
   if Cmodel==1, num=conv(num,Gc_num); den=conv(den,Gc_den); end
   numx=num;
   if Hmodel==1, num=conv(num,H_num); den=conv(den,H_den); end
   denx=[zeros(1,length(den)-length(num)) num]+den;
   if (iplot==1), [mmm,ddd]=bode(num,den,w_range);
   elseif (iplot==6), mmm=step(num,den,t_range);  ddd=step(numx,denx,t_range); end
   shwplot
else, errdisp('Not in Demo'); end
```

可见在系统分析程序 analys.m 中还调用了曲线绘制程序 shwplot.m，该程序的清单为

```
if iplot==1, kbode=kbode+1;
   if kbode==1,
      gbode=figure('pos',[WinGx1 WinGy1, WinGw WinGh],'Name','Bode Diagrams',...
         'NumberTitle','off');
```




```
else, figure(gbode); end
   subplot(211), semilogx(w_range,20*log10(mmm));
   if igrid==1, grid; end, kredraw=0;
   subplot(212), semilogx(w_range,ddd); if igrid==1, grid; end, kredraw=0;
elseif iplot==6, kstep=kstep+1;
   if kstep==1, gstep=figure('pos',[WinGx1-100 WinGy1-100, WinGw WinGh],'Name',...
         'Step Responses','NumberTitle','off');
else, figure(gstep); end
   plot(t_range,[mmm,ddd]); if igrid==1, grid; end, kredraw=0;
else, errdisp('Not in Demo'); end
```

这一程序可以在两个图形窗口上分别绘制出系统的 Bode 图及阶跃响应曲线，事实上在 Control Kit 程序中还允许在不同的图形窗口上同时绘制出各种系统分析曲线。

若用户选择了 Plottings 菜单对图形加以修饰或处理时，则将会自动地调用 pltcom.m 程序，这里的修饰或处理包括的内容如图 8-17(b) 所示，相应的处理程序 pltcom.m 的内容如下

```
if key==1, kstep=0; kbode=0; knyqs=0; knichl=0; krtlc=0; kinvn=0; kimpu=0;
elseif key==2,  plotrng
elseif key==3, glast=gcf; kredraw=1; if igrid==0, igrid=1; else, igrid=0; end, shwplot
elseif key==4, cursor;
elseif key==5, errdisp('Not in Demo'); end
```

这里调用了 plotrng.m 程序，该程序将给出相应的对话框，允许用户改变绘图范围和风格，如改变频率响应分析中的起始频率和终止频率，或阶跃响应中终止时间，并允许用户随意设置图形的点数，还可以设置是否在图形上需要网格线等，这一程序的内容如下

```
gqa=figure('Color',[0;1;1],'pos',[0.25*WinWidth 0.2*WinHeight,0.4*WinWidth 0.3*WinHeight],...
   'Name','Plot Range Define','NumberTitle','off','MenuBar','none');
pos=[0.1 0.7 0.45 0.1]; pos2=[0.62 0.7 0.25 0.1]; k=2;
if (iplot==1)
   namtext='Starting Freq'; vec0='0.1'; getvec;
   pos(2)=pos(2)-0.2; pos2(2)=pos2(2)-0.2; k=3;
   namtext='Ending Freq'; vec0='10'; getvec;
elseif (iplot==6), namtext='Enter terminate time'; vec0='10'; getvec; end
pos(2)=pos(2)-0.2; namtext='Enter number of points';
vec0='150'; k=1; pos2(2)=pos2(2)-0.2; getvec;
itick=uicontrol(gqa,'style','check','string','With Grid?',...
   'position',[0.2 0.05 0.37 0.13],'Units','normalized','Back',[0;1;1]);
mycall=['npoints=val1; igrid=get(itick,''Value''); ',...
   'if iplot==6,tend=val2;t_range=0:tend/npoints:tend; else, w1=log10(val2);',...
   'w2=log10(val3); w_range=logspace(w1,w2,npoints); end, close(gqa); analys'];
uicontrol('Style','Push','Position',[0.7 0.05 0.17 0.13],...
   'Units','normalized','String','OK','Callback',mycall);
```

另外，pltcom.m 程序中还调用了 cursor.m 程序，允许用户在图形上用光标选择曲线点，并在信息窗口 (句柄为 ginfo) 中显示出有关信息。光标处理程序的清单如下

```
figure(glast); [x_pos,y_pos]=ginput(1);
if iplot==1,
   scale=10; [xmin,ii]=min(abs(w_range-x_pos)); var=ddd(ii,:); v=[];
```

```
    for i=1:length(var), v(i)=abs(var(i)-y_pos)*scale/abs(var(i)); end
    [xmin,i]=min(v); if xmin<1, ic=1;  wm=w_range(ii); mm=mmm(ii,i); pm=ddd(ii,i); end
elseif (iplot==6),
    scale=30; [xmin,ii]=min(abs(t_range-x_pos)); var=mmm(ii,:); v=[];
    for i=1:length(var), v(i)=abs((var(i)-y_pos)/var(i)); [xmin,ij]=min(v); xmin=xmin*scale;
end, end
if xmin<1.5,  ic=1; else, ic=0; end
figure(ginfo), clg, pos0=[0.2,0.8,0.5,0.15];
if (iplot<8 & ic==0), shwtext(pos0,col,'Selected point is NOT on the locus ')
elseif (iplot==1),
    shwtext(pos0,col,['The frequency w=', num2str(wm)]);
    pos0(2)=pos0(2)-0.2; shwtext(pos0,col,['The magnitude=', num2str(mm) 'dB']);
    pos0(2)=pos0(2)-0.2; shwtext(pos0,col,['The phase=', num2str(pm) 'deg']);
elseif (iplot==6),
    shwtext(pos0,col,['The time t=' num2str(t_range(ii))]);
    pos0(2)=pos0(2)-0.2; shwtext(pos0,col,['Output y(t)=' num2str(mmm(ii,ij))])
end
figure(glast); hold on, plot(x_pos,y_pos,'o'), hold off
```

简化版本的 Control Kit 软件各个程序或函数的调用关系如图 8-18 所示。

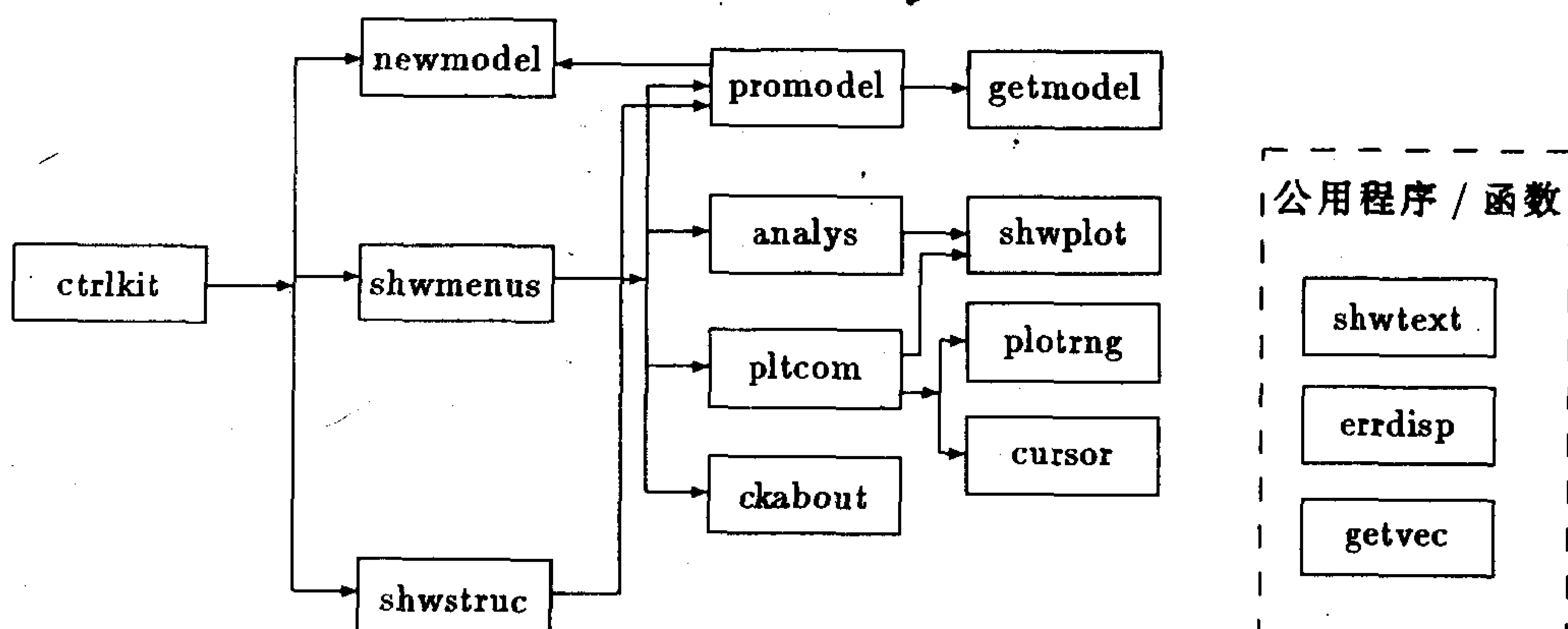


图 8-18　Control Kit 函数调用关系

习　题

1) 开设一个新的图形窗口，使之背景颜色为绿色，并在窗口上保留原有的菜单项。如果想在鼠标器的左键按下之后在命令窗口上显示出 Left Mouse Button Pressed 字样的信息，又将如何编写程序？

2) 在图形窗口上绘制出正弦函数曲线，并设计一个程序段，使得用户用鼠标器在该窗口上拖动出一个方框时，会自动地将被括起来的曲线局部放大。

3) 读下面的程序段

```
h=figure; pos=get(h,'Position');
Colr=[0,0,0; 0,0,1; 0,1,0; 0,1,1; 1,0,0; 1,0,1; 1,1,0; 1,1,1; 0,0,0; 0,0,1];
```



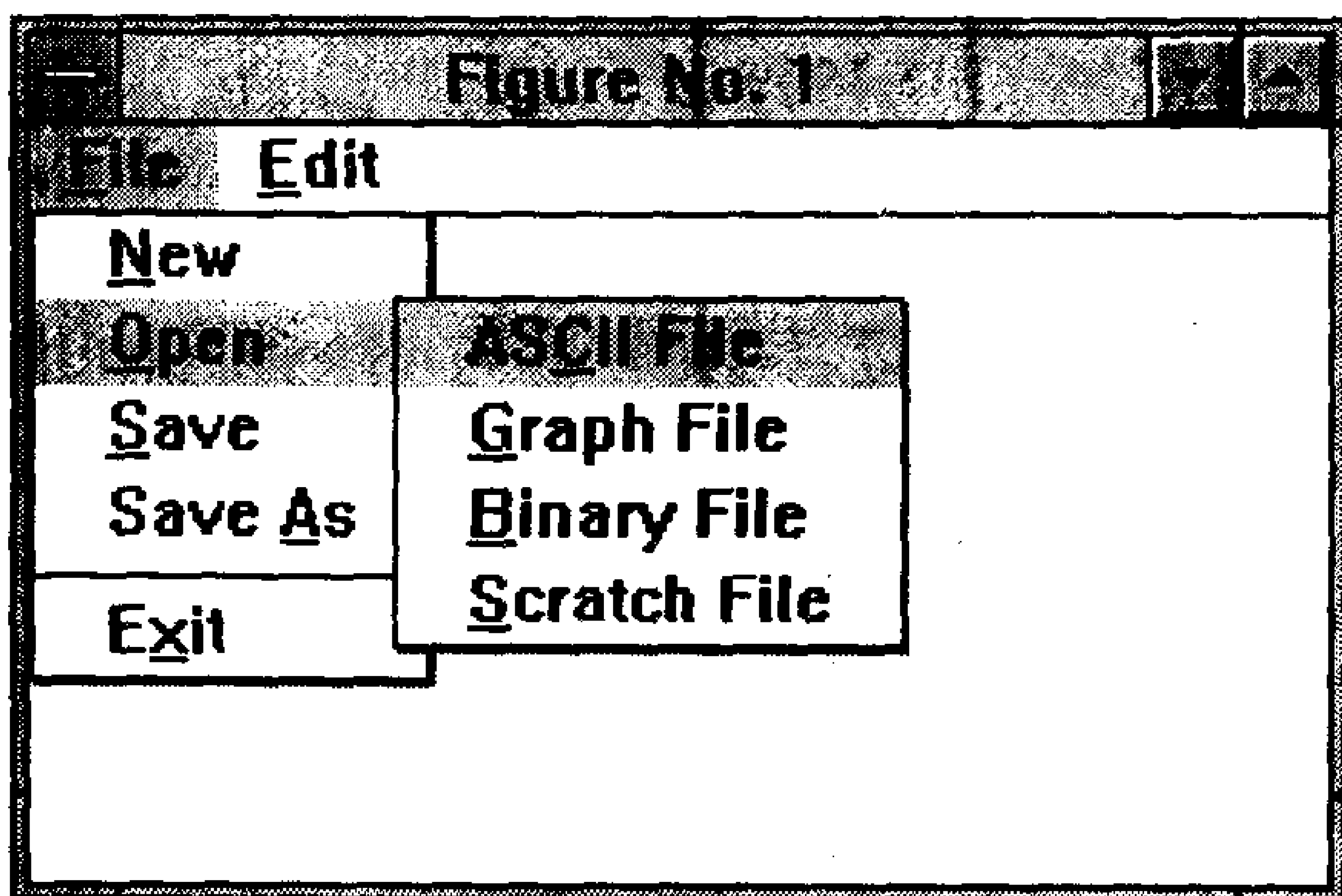

```
for i=1:10
   set(h,'Pos',[pos(1)+10*i, pos(2)-10*i, pos(3:4)], 'Color',Colr(i,:)); pause(2);
end
```

并解释将发生什么结果，在 MATLAB 下运行此程序段对你的预见加以验证，并判断将其中的 pause(2) 语句取消后会发生什么结果，试就此同 MATLAB 实际运行结果进行比较。

4) 在上面的习题中每次改变窗口信息时若想将窗口的标题栏内容改变为 My Plot Windows No i, 其中 i 的字样将随着循环而发生变化，将如何用 MATLAB 语句实现？

5) 设计一个数字电子表型的图形界面，并使之能自动地实时显示时间。

6) 如果想在一个图形窗口上实现下面的菜单系统，将如何设计这样的 MATLAB 程序？若用户已经有过 C 或其它语言的 Windows 编程经验，可以在编程的难易程度上对这两种方法进行比较。



7) 对上面的菜单系统来说，如果用户想在选中了 File | New 选项时打开一个新的窗口，而要打开的窗口具有 1) 中的属性，将如何给出程序实现？

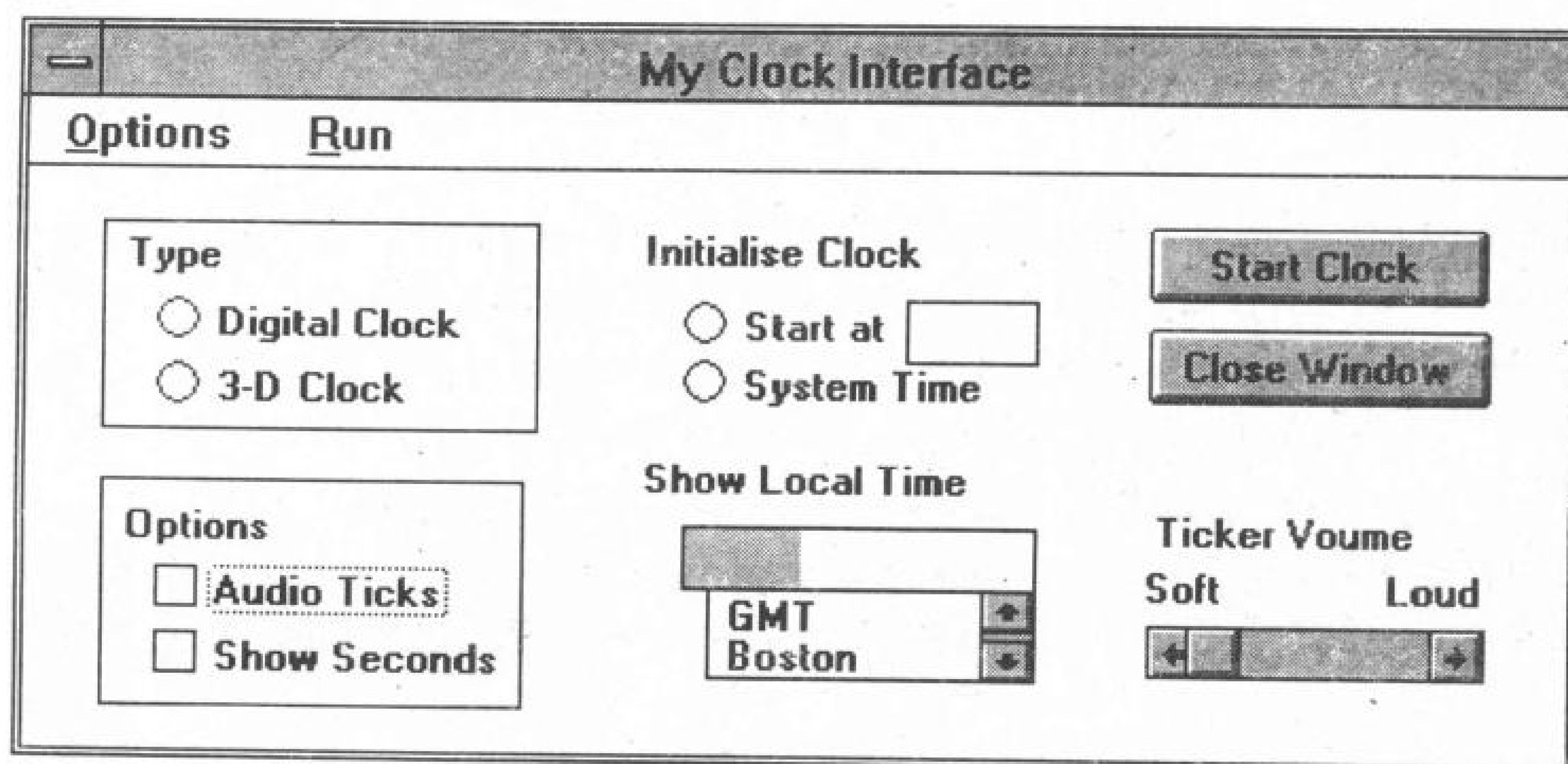
8) 如果想在选中了上面的 File | Append 菜单项，又想在原来的 File 菜单的末尾补足一个标注为 New Item 的子菜单项，使之选中时可以关闭当前的窗口，将如何编写程序？

9) 在使用前面介绍的菜单系统时，如果用户选择了 File | Open 菜单项，则打开一个标准的对话框要求用户读入文件名，输入该文件名之后，可以调用适当的 MATLAB 命令来打开该文件并读入必要的信息，同时在文件名出现错误时会自动给出提示信息。

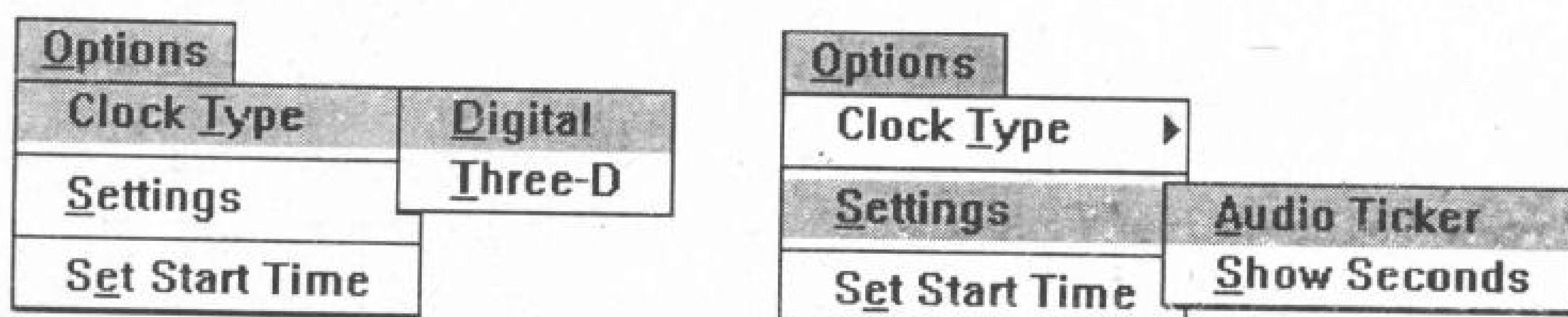
10) 请写出建立下图所示程序界面的 MATLAB 程序，此界面的菜单内容请参见下页的图形。

11) 依照第 6 章介绍的 PID 控制策略及算法，设计一个图形界面程序，允许用户输入系统的模型，并根据模型使用各种方法设计 PID 控制器，并允许对闭环系统进行仿真分析。

12) 利用前面介绍的 Control Kit 程序对 $G(s) = s+1/s(s+2)(s+4)$, $G_c(s) = s+1/0.125s+1$, $H(s)=1$ 构成的反馈控制系统进行分析，如绘制出阶跃响应和 Bode 曲线，并考虑对该程序进行扩充，使得它还可以按照用户的要求绘制出系统的 Nyquist 曲线及 Nichols 图。



习题 10) 附图 (a) 图形界面示意图



习题 10) 附图 (b) 菜单系统表示

13) 给 5.7 节中的非线性频域分析程序设计一个友好的图形界面。

参 考 文 献

[1] Atherton D P, Sarensen O B, Goucem A. Teaching control engineering using implementations of MATLAB. Proceedings Advances in Control Engineering. Japan, 1994

[2] MathWorks. MATLAB™, High-performance numeric computation and visualization software, Building graphical user interfaces, 1993

[3] Xue D, Goucem A, Atherton D P. A menu-driven interface to PC-MATLAB for a first course on control systems. Int. J. of Electrical Engineering Education, 1991, 28(1): 21-33

附录 A MATLAB 函数一览表

函数名	意义	函数名	意义
abs()	绝对值函数	acos()	反余弦函数
acosh()	反双曲余弦	acot()	反余切函数
acoth()	反双曲余切	acsc()	反余割函数
acsch()	反双曲余割	airfoil	NASA 翼面稀疏矩阵显示
all()	测试向量中所有元素是否为真	angle()	相角函数
any()	测试向量中是否有为真元素	ans	返回最新结果
arith	MATLAB 的各种算术运算符信息	asec()	反正割函数
asech()	反双曲正割	asin()	反正弦函数
asinh()	反双曲正弦	atan()	反正切函数
atan2()	四个象限内反正切	atanh()	反双曲正切
auread()	读声音文件	auwrite()	写声音文件
axes()	坐标轴任意形式的设定	axis()	坐标轴标度设定
balance()	改进特征值精度的均衡变换	bar()	条型图曲线绘制
bench	MATLAB 测试基准问题	blanks()	设置一个由空格组成的字符串
bone	带有蓝色的灰度颜色表	break	中断循环执行的语句
brighten()	使图形色调变亮	bucky	Buckminister Fuller 拱形演示
caxis()	伪颜色坐标轴设定	cd	改变当前的工作目录
cdf2rdf()	复块对角矩阵到实块对角阵转换	cedit	设置命令行编辑与回调的参数
ceil()	对 $+\infty$ 方向取整数	census	2000 年美国人口普查预测
chol()	Cholesky 分解	cla	清除当前坐标轴
clabel()	等高线剖面标志	clc	清除命令窗口显示
clear	删除内存中的变量与函数	clf	清除当前图形窗口
clock()	时钟	close()	关闭图形窗口
colmmd()	最小列的阶次	colon	冒号表达式的帮助信息
colormap()	设定颜色可查表	colormenu	颜色表演示
colperm()	由非零数据的计数来排列各列	comet()	彗星状轨迹绘制
comet3()	绘制三维彗星状的轨迹	compan()	生成伴随矩阵

函数名	意义	函数名	意义
compass()	绕行曲线绘制	computer()	计算机类型测试
cond()	求矩阵的条件数	condest()	估算 $\|\cdot\|_1$ 范数
conj()	共轭复数函数	contour()	等高线图形绘制
contour3()	三维等高线绘制	contourc()	等高线绘图计算
contrast()	灰度对比度设置	conv()	求多项式乘法的卷积
conv2()	二维卷积	cool	天蓝粉色基色颜色表
copper	线性铜色调颜色表	corrcoef()	相关函数系数
cos()	余弦函数	cosh()	双曲余弦
cot()	余切函数	coth()	双曲余切
cov()	协方差矩阵	cplxdemo	复变量函数映射函数演示
cplxpair()	将数据按共轭复数对重新排序	cputime()	所用的 CPU 时间
csc()	余割函数	csch()	双曲余割
cumsum()	各元素累加和	cumprod()	各元素累加积
cylinder()	产生柱体	date()	日期
dbclear	清除跟踪调试断点	dbcont	跟踪调试恢复执行
dbdown	改变局部工作空间内容	dbquit	退出跟踪调试模式
dbstack	列出函数调用关系	dbstatus	列出所有的断点情况
dbstep	跟踪调试单步执行	dbstop	设置跟踪调试断点
dbtype	列出带有命令行标号的 .M 文件	dbup	改变局部工作空间内容
deblank()	消除字符串中的空格	dec2hex()	十进制到十六进制的转换
deconv()	因式分解与多项式除法	del2()	五点式离散 Laplace 变换
delete	删除文件	delsqdemo	各种域上的有限差分演示
demo	运行 MATLAB 演示程序	det()	求矩阵的行列式
diag()	建立对角矩阵或获取对角向量	diary	将 MATLAB 运行的命令存盘
diff()	差分函数与近似微分	diffuse()	图像柔焦处理
dir	列出当前目录的内容	disp()	显示矩阵或文本
dmperm()	Dulmage-Mendelsohn 分解	drawnow	闪烁未定的图像事件
earthmap	地球拓扑图形的显示	echo	显示文件中的 MATLAB 命令
eig()	求矩阵的特征值与特征向量	eigmovie	对称矩阵特征值求解过程演示
else	与 if 一起使用的转移语句	elseif	与 if 一起使用的转移语句
end	结束控制语句块的命令	eps	浮点相对误差限 (2.2204×10^{-16})
error()	显示错误信息并中断函数	errorbar()	误差条型图绘制
etree()	矩阵消元树结构	etreeplot()	绘制消元路径
etime()	所用时间的函数	eval()	执行 MATLAB 语句构成的字符串
exist()	检验变量或文件是否已经定义	expm()	矩阵指数函数
expm1()	expm() 函数的 .M 文件实现	expm2()	Taylor 级数法求矩阵指数



函数名	意义	函数名	意义
expm3()	特征值特征向量法求矩阵指数	exp()	指数函数
eye()	产生单位阵	fclose()	关闭文件
feather()	羽状图形绘制	feof()	测试文件是否结束
ferror()	查询文件输入输出错误状态	feval()	执行字符串指定的文件
fft()	离散 Fourier 变换	fft2()	二维离散 Fourier 变换
fftdemo	快速 Fourier 变换演示	fftshift()	取消谱中心零位
fgetl()	从文件读入一行数据（忽略换行）	fgets()	从文件读入一行数据（保留换行）
figure()	生成绘图窗口	fill()	绘制充填的二维多边形
fill3()	绘制充填的三维多边形	filter()	一维数字滤波
filter2()	二维数字滤波	find()	查找非零元素的下标
findstr()	由一个字符串中查找	finite()	若参数为有限元素则为真
fitdemo	非线性最优化拟合演示	fix()	对零方向取整数
flag	红白蓝黑基色颜色表	fliplr()	按左右方向翻转矩阵元素
flipud()	按上下方向翻转矩阵元素	floor()	对 $-\infty$ 方向取整数
flops()	浮点运算计数器	fmin()	单变量最优化函数
fmins()	多变量最优化函数	fopen()	打开文件
for	循环语句	format	设置输出格式
fourier	Fourier 级数展开图形演示	fplot()	给定函数绘图
fplotdemo	函数图形绘制演示	fprintf()	有格式地向文件写入数据，参见 C
fread()	从文件读入二进制数据	frewind()	将文件指针设至文件开头
fscanf()	从文件有格式地读入数据，参见 C	fseek()	设置文件位置指针
ftell()	获得文件位置指针	full()	由稀疏矩阵变换常规矩阵
function	MATLAB 函数表达式的引导符	funm()	矩阵的任意函数
fwrite()	将二进制数据写入文件	fzero()	单变量函数求根
gallery()	生成一些小的测试矩阵	gca()	获得当前坐标轴的句柄
gcf()	获得当前图形的窗口句柄	get()	获得对象属性
getenv	获得环境参数	getframe()	获得一幅“电影”图像
ginput()	由鼠标器作图像输入	global	定义全局变量
gplot()	绘制图论图形	gray	线性灰度颜色表
graymon()	将图形窗口设置成灰度默认值	grid	给图形加网格线
griddata()	插值用数据网格生成	gradient()	近似梯度计算
gtext()	在鼠标指定的位置加文字说明	hadamard()	生成 Hadamard 矩阵
hankel()	生成 Hankel 矩阵	help	启动联机帮助文件显示
hess()	求取 Hessenberg 标准型	hex2num()	十六进制到 IEEE 浮点数的转换
hex2dec()	十六进制到十进制的转换	hidden	网格图隐含线设置开关
hilb()	生成 Hilbert 矩阵	hist()	直方图绘制



函数名	意义	函数名	意义
hold	当前图形保护模式	home	将光标移动到左上角位置
hostid	MATLAB 服务器的主机代号	hot	黑红黄白基色颜色表
hsv	色度饱和值 (HSV) 颜色表	hsv2rgb()	HSV 对 RGB 颜色的转换
if	条件转移语句	ifft()	离散 Fourier 逆变换
ifft2()	二维离散 Fourier 逆变换	imag()	求取虚部函数
image()	创建图像	imagedemo	MATLAB 4.0 版图形处理功能演示
inf	无穷大 (保留变量)	info	显示 MATLAB 与 MathWorks 信息
input()	带有提示的键盘输入函数	int2str()	整数转换为字符串
interp1()	一维插值 (一维查表)	interp2()	二维插值 (二维查表)
interpft()	利用 FFT 的一维插值	intro	MATLAB 引言信息
inv()	矩阵求逆	invhilb()	生成逆 Hilbert 矩阵
isempty()	若参数为空矩阵，则结果为真	isglobal()	若参数为全局变量则为真
ishold()	若屏幕处于保护状态则为真	isieee()	若有 IEEE 算术标准则为真
isinf()	若参数为 Inf，则结果为真	isletter()	若字符串为字母组成则为真
isnan()	若参数为 NaN，则结果为真	issparse()	若矩阵为稀疏表示则为真
isstr()	若参数为字符串，则结果为真	jet	HSV 色调的变化型
keyboard	启动键盘管理程序	knot	围绕三维结的柱形显示
kron()	Kronecker 乘积函数	lasterr()	查询上一条错误信息
length()	查询向量的维数	life	Conway 生命假设的 MATLAB 版
lin2mu()	线性标度文件转换成声音文件	line()	低级折线段绘制函数
linspace()	构造线性分布的向量	load	从文件中读入变量
log()	自然对数函数	log10()	常用对数函数
loglog()	全对数坐标图形绘制	logm()	矩阵的对数
logspace()	构造等对数分布的向量	lookfor	对 HELP 信息中的关键词查找
lorenz	Lorenz 混沌吸引子的曲线	lower()	将一个字符串内容转换为小写
lscov()	最小二乘方差	lu()	矩阵的三角 (LU 分解)
magic()	生成魔术矩阵	matlabrc	启动主程序
max()	求向量中最大元素	mean()	求向量各元素均值
median()	求向量各元素中间值	membrane	产生 MathWorks 公司标志
menu()	产生用户输入的菜单	mesh()	三维网格图形
meshc()	带有等高线的网格图形	meshgrid()	构造三维图形用 x,y 阵列
meshz()	带有零平面的三维网格图形	min()	求向量中最小元素
more	控制命令窗口的输出页面	movie()	播放存储的“电影”幅面
moviein()	初始化“电影”各幅图像内存	mu2lin()	声音文件对线性标度文件的转换
NaN	不定式	nargchk()	函数输入输出参数个数检验
nargin	函数中实际输入变量个数	nargout	函数中实际输出变量个数



函数名	意义	函数名	意义
newplot()	NextPlot 特性的 .M 文件前缀	nextpow2()	找出下一个 2 的指数
nnls()	非零最小二乘	nnz()	非零元素个数
nonzeros()	非零元素	norm()	求矩阵的范数
normest()	估算 $\|\cdot\|_2$ 范数	null()	右零空间
num2str()	将数值转换为字符串	nzmax()	允许的非零元素存储空间
ode23()	微分方程低阶数值解法	ode23p()	微分方程低阶数值解法并画图
ode45()	微分方程高阶数值解法	odedemo	常微分方程演示
ones()	产生元素全部为 1 的矩阵	orient()	设置打印纸方向
orth()	正交空间	pack	整理工作空间内存
patch()	低级填充多边形绘制函数	path	设置或查询 MATLAB 的路径
paren	各种括号的查询信息	pascal()	生成 Pascal 矩阵
pause()	暂停函数	pcolor()	伪颜色绘图
peaks	两变量的峰值函数演示	penny	便士硬币的各个角度视图
pi	圆周率 π	pink	粉色色调颜色表
pinv()	伪逆矩阵	plot()	线性坐标图形绘制
plot3()	绘制三维线或点型图形	polar()	极坐标图形绘制
poly()	求矩阵的特征多项式	polyder()	多项式求导
polyfit()	数据的多项式拟合	polyval()	多项式求值
polyvalm()	多项式矩阵求值	print()	打印图形或将图形存盘
printopt()	建立打印机默认值	prism	光谱颜色表
prod()	对向量中各元素求积	punct	各种标点符号的查询信息
qr()	矩阵的正交三角化 (QR) 分解	qrdelete()	QR 分解中删除一列
qrinsert()	QR 分解中插入一列	quad()	低阶数值积分算法
quad8()	高阶数值积分算法	quaddemo	自适应变步长数值积分演示
quake	Loma Prieta 地震模型	quit	退出 MATLAB 环境
quiver()	箭头图形	qz()	广义特征值问题求解 (QZ 算法)
rand()	产生随机矩阵	randn()	产生正态分布随机阵
randperm()	随机置换向量	rank()	求矩阵的秩
rbbox()	擦除框	rcond()	LINPACK 倒数条件估计
real()	求取实部函数	realmax	最大浮点数值
realmin	最小浮点数值	relop	各种关系符号的查询信息
rem()	除法的余数	reset()	恢复对象特性
reshape()	改变矩阵的行列数目	residue()	部分分式展开
return	返回到主调函数的命令	rgb2hsv()	RGB 对 HSV 颜色的转换
rgbplot()	绘制颜色图	roots()	求多项式的根
rose()	极坐标（角度）直方图绘制	rosser()	典型的对称矩阵特征值问题测试

函数名	意义	函数名	意义
rot90()	将矩阵元素旋转 90 度	round()	截取到最近的整数
rref()	矩阵的行阶梯型实现	rrefmovie	消元法解方程过程演示
rsf2csf()	实块对角阵转换复块对角阵	save	将工作空间中变量存盘
saxis()	声音坐标轴处理	surfnorm()	表面图形规范化
schur()	Schur 分解	script	MATLAB 语句及文件信息
sec()	正割函数	sech()	双曲正割
semilogx()	x 轴半对数坐标图形绘制	semilogy()	y 轴半对数坐标图形绘制
sepdemo	有限元网格图演示	set()	设置对象属性
setstr()	将数值转换为字符串	shading	阴影模式
sigdemo1	离散 Fourier 变换演示	sigdemo2	连续 Fourier 变换演示
sign()	符号函数	sin()	正弦函数
sinh()	双曲正弦	size()	查询矩阵的维数
slash	求解线性方程（左除右除）信息	slice()	容量可视图形
sort()	对向量中各元素排序	sound()	将数据向量转换为声音
sounddemo	MATLAB 4.0 的声音功能演示	spalloc()	给非零元素定位存储空间
sparse()	从常规矩阵转换稀疏矩阵	sparsity	稀疏矩阵排序效应演示
spaugment()	建立最小二乘增广系统	spconvert()	由稀疏矩阵外部格式进行转换
spdiags()	稀疏对角矩阵	specular()	反射
speye()	稀疏单位矩阵	spfun()	对稀疏矩阵处理的非线性函数
sphere()	产生球面	spinmap()	使颜色旋转
spline2d	二维样条函数演示	spones()	将原稀疏矩阵非零元素用 1 取代
spparms()	设置稀疏矩阵参数	sprank()	结构秩数
sprandn()	稀疏随机矩阵	sprandsym()	稀疏对称随机矩阵
sprintf()	按照 C 语言格式书写字符串	spy()	绘制稀疏矩阵结构
sqdemo	超二次锥面的显示	sqrt()	平方根函数
sqrtm()	矩阵的平方根	sscanf()	按照 C 语言格式读字符串
stairs()	阶梯图形绘制	startup	MATLAB 自启动文件
std()	求向量中各元素标准方差	stem()	离散序列柄状图形绘制
str2mat()	字符串转换成矩阵	str2num()	字符串转换为实型数据
strcmp()	字符串比较	strings	关于 MATLAB 字符串的帮助信息
subplot()	将图形窗口分成若干个区域	subscribe	成为 MATLAB 的签约用户
subspace()	子空间	sum()	对向量中各元素求和
sunspots	太阳黑子活动模拟	surf()	三维表面图形
surface()	创建曲面	surfc()	带有等高线的三维表面图形
surfl()	带有光照阴影的三维表面图形	svd()	奇异值分解 (SVD)
symbfact()	符号因式分解	symmmd()	对称最小阶次



函　数　名	意　　义	函　数　名	意　　义
symrcm()	逆序 Cuthill-McKee 排序	tan()	正切函数
tanh()	双曲正切	terminal	设置图形终端类型
text()	在图形上加文字说明	tic()	启动秒表计时器
title()	给图形加标题	toc()	读取秒表计时器
toeplitz()	生成 Toeplitz 矩阵	trace()	求矩阵的迹
trapz()	梯形法求数值积分	treelayout()	树状结构
treeplot()	画出分割路径的图形	tril()	提取矩阵的下三角部分
triu()	提取矩阵的上三角部分	type	列出 .M 文件
uicontrol()	建立用户界面控制的函数	uigetfile()	标准读盘文件名处理对话框
uimenu()	建立界面的菜单	uiputfile()	标准存盘文件名处理对话框
uisetcolor()	标准颜色设置对话框	uisetfont()	标准字体设置对话框
unix	执行操作系统命令并返回结果	unwrap()	除去每 360° 的跳跃
upper()	将一个字符串内容转换为大写	vander()	生成 Vandermonde 矩阵
ver	显示程序版本号	version	显示 MATLAB 版本号
vibes	L 型振荡动画	view()	三维图形视口指定
viewmtx()	显示坐标变换矩阵	waterfall()	瀑布型图形
what	列出当前目录下的有关文件	whatsnew	手册中未给出的新特性
which	找出函数与文件所在的目录名	while	循环语句
whitebg	将图形窗口设置成白色背景	who	简要列出工作空间变量名
whos	详细列出工作空间变量名	why	给出简要的回答
wilkinson()	生成 Wilkinson 特征值测试矩阵	xlabel()	给图形加 X 坐标说明
xor()	逻辑异或	ylabel()	给图形加 Y 坐标说明
zerodemo	求根演示	zeros()	产生零矩阵
zlabel()	给图形加 Y 坐标说明		



附录 B MATLAB 函数分类索引

- **通用函数与命令:**

  1) 管理用命令: help, what, type, lookfor, which, demo, path

  2) 变量与工作空间的管理: who, whos, load, save, clear, pack, size(), length(), disp()

  3) 文件和操作系统的处理 : cd, dir, delete, getenv, !, unix, diary

  4) 命令窗口控制: cedit, clc, home, format, echo, more

  5) 启动与退出 MATLAB: quit, startup, matlabrc

  6) 一般信息: info, subscribe, hostid, whatsnew, ver

- **运算符号与特殊字符**

  1) 运算符号与特殊字符: +, −, *, .*, , ., \, /, ./, kron(), :, (,), [,], ., .., ..., , ; %, !, ', =, ==, <, >, &, |, ~, xor()

  2) 逻辑特性函数: exist(), any(), all, find(), isnan(), isinf(), isinfite(), isempty(), issparse(), isstr(), isglobal()

- **语言结构与跟踪调试**

  1) 编程语言用 MATLAB 结构: script, function, eval(), feval(), global, nargchk(), lasterr(),

  2) 控制流程: if, else, elseif, end, for, while, break, return, error

  3) 交互输入函数及命令: input(), keyboard, menu(), pause, uimenu(), uicontrol()

  4) 跟踪调试命令 : dbstop, dbclear, dbcont, dbdown, dbstack, dbstatus, dbstep, dbtype, dbup, dbquit

- **基本矩阵与矩阵处理**

  1) 基本矩阵: zeros(), ones(), eye(), rand(), randn(), linspace(), logspace(), meshgrid(), :

  2) 特殊变量与常量: ans, eps, realmax, realmin, pi, i, j, inf, NaN, flops(), nargin, nargout, computer, isieee(), why, version

  3) 时间与日期: clock, cputime(), date(), etime(), tic(), toc()

  4) 矩阵处理: diag(), fliplr(), flipud(), reshape(), rot90(), tril(), triu(),

- **矩阵函数及数值线性代数**

1) 矩阵分析: cond(), norm(), rcond(), rank(), det(), trace(), null(), orth(), rref()

2) 线性方程: \, /, chol(), lu(), inv(), qr(), qrdelete(), qrinsert(), nnls(), pinv(), lscov()

3) 特征值与奇异值: eig(), poly(), hess(), qz(), rsf2csf(), cdf2rdf(), schur(), balance(), svd()

4) 矩阵函数: expm(), expm1(), expm2(), expm3(), logm(), sqrtm(), funm()

5) 特殊矩阵: compan(), gallery(), hadamard(), hankel(), hilb(), invhilb(), kron(), magic(), pascal(), rosser(), toeplitz(), vander(), wilkinson()

- **基本数学函数**

1) 三角函数: sin(), sinh(), asin(), asinh(), cos(), cosh(), acos(), acosh(), tan(), tanh(), atan(), atan2(), atanh(), sec(), sech(), asec(), asech(), csc(), csch(), acsc(), acsch(), cot(), coth(), acot(), acoth()

2) 指数函数: exp(), log(), log10(), sqrt()

3) 复数函数: abs(), angle(), conj(), imag(), real()

4) 数值处理: fix(), floor(), ceil(), round(), rem(), sign()

- **数据分析与 Fourier 变换函数**

1) 基本运算: max(), min(), mean(), median(), std(), sort(), sum(), prod(), cumsum(), cumprod(), trapz(),

2) 微分运算: diff(), gradient(), del2(),

3) 方差处理: corrcoef(), cov(), subspace()

4) 滤波和卷积: filter(), filter2(), conv(), conv2(), deconv()

5) Fourier 变换: fft(),fft2(), ifft(), ifft2(), abs(), angle(), unwrap(), fftshift(), cplxpair(), nextpow2()

- **多项式处理函数**

1) 多项式处理: roots(), poly(), polyval(), polyvalm(), residue(), polyfit(), polyder(), conv(), deconv()

2) 数据插值: interp1(), interp2(), interpft(), griddata()

- **非线性数值方法**: ode23(), ode23p(), ode45(), quad(), quad8(), fmin(), fmins(), fzero(), fplot()
- **稀疏矩阵函数**

1) 基本稀疏矩阵: speye(), sprandn(), sprandsym(), spdiags()diagonals.

2) 稀疏矩阵转换: sparse(), full(), find(), spconvert()

3) 处理零元素: nnz(), nonzeros(), nzmax(), spones(), spalloc(), issparse(), spfun()

4) 稀疏矩阵可视化: spy(), gplot()

5) 重新排序算法: colmmd(), symmmd(), symrcm(), colperm(), randperm(), dmperm()

6) 范数条件数等参数: normest(), condest(), sprank()

7) 树结构操作： treelayout(), treeplot(), etree(), etreeplot()

8) 其它函数： symbfact(), spparms()、spaugment()

- **二维图形绘制**

1) 基本二维图形：plot(), loglog(), semilogx(), semilogy(), fill()

2) 特殊二维图形：polar(), bar(), stem(), stairs(), errorbar(), hist(), rose(), compass(), feather(), fplot(), comet()

3) 图形修饰：title(), xlabel(), ylabel(), text(), gtext(), grid

- **三维图形绘制**

1) 直线与充填： plot3(), fill3(), comet3()

2) 等高线和其它二维图形：contour(), contour3(), clabel(), contourc(), pcolor(), quiver()

3) 表面与网络图形：mesh(), meshc(), meshz(), surf(), surfc(), surfl(), waterfall() , slice()

4) 图形外观处理： view(), viewmtx(), hidden, shading, axis(), caxis(), colormap()

5) 三维对象： cylinder(), sphere()

- **图形处理**

1) 图形窗口操作和控制：figure(), gcf, clf, close()

2) 坐标轴建立与控制：subplot(), axes(), gca, cla, axis(), caxis(), hold

3) 处理图形对象： figure(), axes(), line(), text(), patch(), surface(), image(), uicontrol(), uimenu()

4) 处理图形运算： set(), get(), reset(), delete, drawnow, newplot

5) 硬拷贝与存储： print, printopt, orient

6) 动画处理： moviein(), getframe(), movie()

7) 其它概念： ginput(), rbbox(), ishold(), whitebg, graymon(), terminal, uiputfile(), uigetfile(), uisetcolor(), uisetfont()

- **颜色与光照模型处理**

1) 颜色控制：colormap(), caxis(), shading

2) 色调设置：hsv(), gray(), hot(), cool(), bone(), copper(), pink(), prism(), jet(), flag()

3) 有关于色调的函数： hsv2rgb(), rgb2hsv contrast(), brighten(), spinmap(), rgbplot()

4) 光照模型：surfl(), specular(), diffuse(), surfnorm()

- **声音处理**

1) 基本声音函数： sound(), saxis()

2) 计算机用声音函数：auwrite()，auread()，mu2lin()，lin2mu()

- **字符串处理函数**

1) 基本处理：strings()，abs()，setstr()，isstr()，blanks()，deblank()，str2mat()，eval()

2) 字符串比较：strcmp()，findstr()，upper()，lower()，isletter()

3) 字符串与数值转换：num2str()，int2str()，str2num()，sprintf()，sscanf()

4) 十六进制转换：hex2num()，hex2dec()，dec2hex()

- **低级输入输出函数**

1) 文件打开与关闭：fopen()，fclose()

2) 无格式输入输出：fread()，fwrite()

3) 有格式输入输出：fscanf()，fprintf()，fgetl()，fgets()

4) 文件定位：ferror()，feof()，fseek()，ftell()，frewind()

5) 字符串转换：sprintf()，sscanf()

- **演示程序**

1) 入门程序：demo，intro，bench

2) MathWorks 特别演示：vibes，life，penny，lorenz，earthmap，sqdemo

3) 数据方向演示：fftdemo，quake，sigdemo1，sigdemo2，census，spline2d，sunspots

4) 数值分析演示：odedemo，quaddemo，zerodemo，fplotdemo，fitdemo，eigmovie，rrefmovie

5) 数学问题演示：fourier，cplxdemo，peaks，knot，membrane

6) 图像与色调及声音演示：imagedemo，colormenu，sounddemo

7) 稀疏矩阵演示：bucky，delsqdemo，sepdemo，sparsity，airfoil